```
DDDDDDDDDDD     EEEEEEEEEEEEEEE  BBBBBBBBBBB      UUU         UUU     GGGGGGGGGGG
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 EEEEEEEEEEE      BBBBBBBBBBBB     UUU         UUU GGG
DDD         DDD EEEEEEEEEEE      BBBBBBBBBBBB     UUU         UUU GGG
DDD         DDD EEEEEEEEEEE      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      UUUUUUUUUUUUUUUUU     GGGGGGGGG
DDDDDDDDDDDD    EEEEEEEEEEEEEEE  BBBBBBBBBBBB      UUUUUUUUUUUUUUUUU     GGGGGGGGG
DDDDDDDDDDDD    EEEEEEEEEEEEEEE  BBBBBBBBBBBB      UUUUUUUUUUUUUUUUU     GGGGGGGGG
```

```
DDDDDDDD    BBBBBBBB      GGGGGGGG  PPPPPPPP      AAAAAA    RRRRRRRR      SSSSSSSS  EEEEEEEEEE  RRRRRRRR
DDDDDDDD    BBBBBBBB      GGGGGGGG  PPPPPPPP      AAAAAA    RRRRRRRR      SSSSSSSS  EEEEEEEEEE  RRRRRRRR
DD      DD  BB      BB  GG          PP      PP  AA      AA  RR      RR  SS          EE          RR      RR
DD      DD  BB      BB  GG          PP      PP  AA      AA  RR      RR  SS          EE          RR      RR
DD      DD  BB      BB  GG          PP      PP  AA      AA  RR      RR  SS          EE          RR      RR
DD      DD  BB      BB  GG          PP      PP  AA      AA  RR      RR  SS          EE          RR      RR
DD      DD  BBBBBBBB    GG          PPPPPPPP    AA      AA  RRRRRRRR      SSSSSS    EEEEEEEE    RRRRRRRR
DD      DD  BBBBBBBB    GG          PPPPPPPP    AA      AA  RRRRRRRR      SSSSSS    EEEEEEEE    RRRRRRRR
DD      DD  BB      BB  GG  GGGGGG  PP          AAAAAAAAAA  RR  RR              SS  EE          RR  RR
DD      DD  BB      BB  GG  GGGGGG  PP          AAAAAAAAAA  RR  RR              SS  EE          RR  RR
DD      DD  BB      BB  GG      GG  PP          AA      AA  RR    RR            SS  EE          RR    RR
DD      DD  BB      BB  GG      GG  PP          AA      AA  RR    RR            SS  EE          RR    RR
DDDDDDDD    BBBBBBBB      GGGGGG    PP          AA      AA  RR      RR  SSSSSSSS    EEEEEEEEEE  RR      RR
DDDDDDDD    BBBBBBBB      GGGGGG    PP          AA      AA  RR      RR  SSSSSSSS    EEEEEEEEEE  RR      RR

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

```
;     1          0001  0 MODULE DBGPARSER (IDENT = 'V04-000') =
;     2          0002  0
;     3          0003  1 BEGIN
;     4          0004  1
;     5          0005  1 !**************************************************************************
;     6          0006  1 !*                                                                        *
;     7          0007  1 !*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                              *
;     8          0008  1 !*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                *
;     9          0009  1 !*  ALL RIGHTS RESERVED.                                                  *
;    10          0010  1 !*                                                                        *
;    11          0011  1 !*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED  *
;    12          0012  1 !*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE  *
;    13          0013  1 !*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER  *
;    14          0014  1 !*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY  *
;    15          0015  1 !*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY  *
;    16          0016  1 !*  TRANSFERRED.                                                           *
;    17          0017  1 !*                                                                        *
;    18          0018  1 !*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE  *
;    19          0019  1 !*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT  *
;    20          0020  1 !*  CORPORATION.                                                           *
;    21          0021  1 !*                                                                        *
;    22          0022  1 !*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS  *
;    23          0023  1 !*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                *
;    24          0024  1 !*                                                                        *
;    25          0025  1 !*                                                                        *
;    26          0026  1 !**************************************************************************
;    27          0027  1 !
;    28          0028  1 ! WRITTEN BY
;    29          0029  1 !       Bert Beander    February, 1982
;    30          0030  1 !
;    31          0031  1 ! MODIFIED BY
;    32          0032  1 !       Rich Title                      Added code and tables needed to
;    33          0033  1 !                                       support BLISS, C, MACRO.
;    34          0034  1 !       Ping Sager                      Added code and tables needed to
;    35          0035  1 !                                       support PASCAL.
;    36          0036  1 !       Walter Carrell III              Update DBG$GL_CURRENT_PRIMARY for
;    37          0037  1 !                                       self_referential records
;    38          0038  1 !       Brad Becker                     Added tables and support for
;    39          0039  1 !                                       Built-in functions.
;    40          0040  1 !
;    41          0041  1 !
;    42          0042  1 ! MODULE FUNCTION
;    43          0043  1 !       This module contains the language-independent lexical scanner and
;    44          0044  1 !       parser used to scan and parse both address expressions and language
;    45          0045  1 !       expressions for all languages.  It also contains all the parse tables
;    46          0046  1 !       needed to scan and parse each of the languages supported by DEBUG.
;    47          0047  1 !
;    48          0048  1
;    49          0049  1 REQUIRE 'SRC$:DBGPROLOG.REQ';
;    50          0183  1
;    51          0184  1 LIBRARY 'LIB$:DBGGEN.L32';
;    52          0185  1
;    53          0186  1 FORWARD ROUTINE
;    54          0187  1     AAA_DUMMY: NOVALUE,                 ! Dummy routine--does nothing, not used
;    55          0188  1     DBG$ADDR_EXP_INT,                   ! Address Expression Interpreter driver
;    56          0189  1     DBG$BUILD_PRIMARY_SUBNODE: NOVALUE, ! Build a Primary Descriptor Sub-Node
;    57          0190  1     DBG$EXP_INT,                        ! Expression Interpreter driver routine
```

```
;    58          0191  1     DBG$EXPRESSION_PARSER,                ! Parser to parse language expressions
;    59          0192  1     DBG$GET_BIF_ARGUMENTS,                ! Parse and pick up built-in function argument list
;    60          0193  1     DBG$LEXICAL_SCANNER,                  ! Lexical scanner for all languages
;    61          0194  1     DBG$PARSER_SET_LANGUAGE: NOVALUE,     ! Set language tables for parser
;    62          0195  1     DBG$PRIMARY_PARSER: NOVALUE,          ! Parser to parse a primary symbol
;    63          0196  1     APPEND_TO_PATHNAME: NOVALUE,          ! Append a name to Pathname Descriptor
;    64          0197  1     CHECK_OPSCOPE,                        ! Go upscope from record component
;    65          0198  1     CONSTANT_TO_VALDESCR,                 ! Build Value Descriptor for a constant
;    66          0199  1     CREATE_OPERAND_TOKEN,                 ! Create a Token Entry for an operand
;    67          0200  1     CREATE_OPERATOR_TOKEN,                ! Create a Token Entry for an operator
;    68          0201  1     CREATE_PRID_CONSTANT,                 ! Create a Predefined Identifier Constatn
;    69          0202  1                                           !       value descriptor
;    70          0203  1     DUMP_OPERATOR: NOVALUE,               ! Dump an operator being evaluated
;    71          0204  1     DUMP_TOKEN: NOVALUE,                  ! Dump a Token Entry symbolically
;    72          0205  1     DUMP_PRIMARY: NOVALUE,                ! Dump a Primary or Value Descriptor
;    73          0206  1     FIX_UP_PRIMARY: NOVALUE,              ! Fix up a Primary with subscripts
;    74          0207  1     GET_BLISS_SUBSCRIPTS: NOVALUE,        ! Pick up BLISS subscripts
;    75          0208  1     GET_DEREFERENCE: NOVALUE,             ! Do a derefence (PASCAL ^ operator)
;    76          0209  1     GET_FIELDREF: NOVALUE,                ! Pick up field reference X<p,s,e>
;    77          0210  1     GET_RECORD_COMPONENT: NOVALUE,        ! Do record component selection
;    78          0211  1     GET_RECORD_VARIANT,                   ! Search a set of record variants for
;    79          0212  1                                           !       a specified record component
;    80          0213  1     GET_SET_CONSTANT,                     ! Parse and pick of set constants
;    81          0214  1     GET_SUBSCRIPTS: NOVALUE,              ! Parse and pick of array subscripts
;    82          0215  1     GET_SUBSTRING: NOVALUE,               ! Parse and pick up substring reference
;    83          0216  1     OPERATOR_TO_RESTORE_RADIX,            ! Returns operator to restore radix
;    84          0217  1     PATHNAME_TO_PRIMARY,                  ! Construct Primary Descr from pathname
;    85          0218  1     RESOLVE_COMPONENT,                    ! Attempt to resolve possibly
;    86          0219  1                                           !    ambiguous record references.
;    87          0220  1     SAVE_SUBSCRIPTS: NOVALUE,             ! Save away subscripts
;    88          0221  1     SCAN_QUOTED_STRING: NOVALUE;          ! Scan a quoted character string
```

```
;    90          0222  1 EXTERNAL ROUTINE
;    91          0223  1     DBG$DATA_LENGTH,                    ! Length in bits of data in VMS descriptor
;    92          0224  1     DBG$DEF_SYM_FIND,                   ! Look up DEFINEd symbol
;    93          0225  1     DBG$DUMP_HEX: NOVALUE,              ! Dump a memory block in hexadecimal
;    94          0226  1     DBG$ENUM_POS,                       ! Convert enum val-> enum position
;    95          0227  1     DBG$ENUM_VAL,                       ! Convert enum pos-> enum val
;    96          0228  1     DBG$EVALOP_SET_LANGUAGE: NOVALUE,   ! Sets up Operator Information Tables
;    97          0229  1     DBG$EVAL_ADA_TICK,                  ! Evaluate an Ada Tick operator
;    98          0230  1     DBG$EVAL_ADDR_OPERATOR,             ! Evaluate an Address Expr. operator
;    99          0231  1     DBG$EVAL_LANG_OPERATOR,             ! Evaluate a language expr. operator
;   100          0232  1     DBG$GET_TEMPMEM,                    ! Get a temporary memory block
;   101          0233  1     DBG$HASH_FIND,                      ! Look up symbol in RST Hash Table
;   102          0234  1     DBG$HASH_FIND_SETUP: NOVALUE,       ! Set up call on DBG$HASH_FIND
;   103          0235  1     DBG$PERFORM_TYPEID_CHECK,           ! Check type consistency in set constant
;   104          0236  1     DBG$MAKE_SKELETON_DESC,             ! Make skeleton Value or Primary Descr.
;   105          0237  1     DBG$MAP_DTYPE_CLASS,                ! Given DTYPE, do a best guess at the CLASS
;   106          0238  1     DBG$NCOB_PATHDESC_TO_CS: NOVALUE,   ! Pathname to COBOL string
;   107          0239  1     DBG$NCOPY_DESC,                     ! Copy a descriptor
;   108          0240  1     DBG$NEWLINE: NOVALUE,               ! Close print line and start new line
;   109          0241  1     DBG$NPATHDESC_TO_CS: NOVALUE,       ! Convert Pathname Descr to ASCII
;   110          0242  1     DBG$NUM_BYTES,                      ! Return the length of a given dtype
;   111          0243  1     DBG$PRIM_TO_ADDR,                   ! Convert Primary to address
;   112          0244  1     DBG$PRIM_TO_VAL,                    ! Convert Primary to Value Descriptor
;   113          0245  1     DBG$PRINT: NOVALUE,                 ! Print FAO-formatted text
;   114          0246  1     DBG$PRINT_SET_LANGUAGE: NOVALUE,    ! Sets up Print Information Tables
;   115          0247  1     DBG$STA_GETSYMBOL: NOVALUE,         ! Look up a symbol in the RST
;   116          0248  1     DBG$STA_SETCONTEXT: NOVALUE,        ! Set up current context
;   117          0249  1     DBG$STA_SYMSIZE: NOVALUE,           ! Look up a symbol's size
;   118          0250  1     DBG$STA_SYMTYPE: NOVALUE,           ! Look up a symbol's data type in RST
;   119          0251  1     DBG$STA_SYMVALUE: NOVALUE,          ! Look up a symbol's value
;   120          0252  1     DBG$STA_TYPEFCODE,                  ! Get FCODE of a given Type ID
;   121          0253  1     DBG$STA_TYP_ARRAY: NOVALUE,         ! Look up array data type information
;   122          0254  1     DBG$STA_TYP_ATOMIC: NOVALUE,        ! Look up atomic data type information
;   123          0255  1     DBG$STA_TYP_DESCR: NOVALUE,         ! Look up descriptor type information
;   124          0256  1     DBG$STA_TYP_FILE: NOVALUE,          ! Get type info for file variable
;   125          0257  1     DBG$STA_TYP_RECORD: NOVALUE,        ! Look up record data type information
;   126          0258  1     DBG$STA_TYP_SUBRNG: NOVALUE,        ! Look up subrange data type information
;   127          0259  1     DBG$STA_TYP_TYPEDPTR: NOVALUE,      ! Look up typed pointer information
;   128          0260  1     DBG$STA_TYP_VARIANT: NOVALUE,       ! Look up record variant set information
;   129          0261  1     DBG$STA_TYP_VARIANT_COMP: NOVALUE,  ! Look up components of a record variant
;   130          0262  1     DBG$TYPEID_FOR_ARRAY,               ! Construct a typeid for an array
;   131          0263  1     DBG$TYPEID_FOR_ATOMIC,              ! Construct a typeid for an atomic object
;   132          0264  1     DBG$TYPEID_FOR_SET,                 ! Construct a Set Constant typeid
;   133          0265  1     OTS$CVT_TI_L;                       ! Convert ASCII string to integer
;   134          0266  1
;   135          0267  1 EXTERNAL
;   136          0268  1     DBG$GL_ARRSUB_FLAG,
;   137          0269  1     DBG$GL_DEVELOPER: BITVECTOR[],      ! Developer mode control bits
;   138          0270  1     DBG$GL_CURRENT_PRIMARY,             ! Pointer to the primary being processed
;   139          0271  1     DBG$GB_LANGUAGE: BYTE,              ! Currently set language
;   140          0272  1     DBG$GB_MOD_PTR: REF VECTOR [,BYTE], ! Pointer to modes
;   141          0273  1     DBG$GL_ORIG_COMMAND_PTR,            ! Pointer to original command string
;   142          0274  1     DBG$GL_RECCMP_FLAG,
;   143          0275  1     DBG$GB_SET_BREAK_FLAG: BYTE,        ! Set to TRUE during parsing of
;   144          0276  1                                         !    SET BREAK command.
;   145          0277  1     DBG$GL_UPCASE_COMMAND_PTR: VECTOR[2];
;   146          0278  1                                         ! Pointers to start and end
```

```
;  147     0279  1                                        !    of current command string
;  148     0280  1
;  149     0281  1 GLOBAL
;  150     0282  1     DBG$GL_CHARTBL: REF CHRTBL$TABLE;   ! Pointer to character table for the
;  151     0283  1                                         !      currently set language
;  152     0284  1
;  153     0285  1 LITERAL
;  154     0286  1     CAR_RET = %CHAR(13),                ! Carriage-return character
;  155     0287  1     VARSTK_SIZE = 20;                   ! Maximum nesting depth of record
;  156     0288  1                                         !      variants we allow
;  157     0289  1
;  158     0290  1 ! These are passed into APPEND_TO_PATHNAME as the third parameter. This
;  159     0291  1 ! parameter tells the routine how to insert the name into the pathname
;  160     0292  1 ! being constructed.
;  161     0293  1 !
;  162     0294  1 LITERAL
;  163     0295  1     NOT_REC_COMP        = 0,
;  164     0296  1     REC_COMP            = 1,
;  165     0297  1     COB_REC_COMP        = 2;
;  166     0298  1
;  167     0299  1 OWN
;  168     0300  1     ADDRESS_LENGTH,                     ! Length of instruction pointed to by
;  169     0301  1                                         !      address expression (or zero)
;  170     0302  1     ADDRESS_TYPE,                       ! Address expression "type"--either
;  171     0303  1                                         !      "instruction" or "no type"
;  172     0304  1     BIF_TABLE: REF VECTOR[,LONG],       ! Pointer to a Built-in Function table
;  173     0305  1                                         !      for the current language where
;  174     0306  1                                         !      the functions are identifiers.
;  175     0307  1                                         !      (SUCC, PRED)
;  176     0308  1     CASING_SIGNIFICANT,                 ! Set to TRUE if upper/lower casing
;  177     0309  1                                         !      is significant in names.
;  178     0310  1     CHARPTR: REF VECTOR[,BYTE],         ! Pointer to current character in input
;  179     0311  1                                         !      line being scanned and parsed
;  180     0312  1     CHARTBL: CHRTBL$TABLE,              ! Character Table for current language
;  181     0313  1     COMPONENTS_IN_PATHNAME,             ! Set to TRUE if the current language
;  182     0314  1                                         !      picks up all record components
;  183     0315  1                                         !      before calling GETSYMBOL.
;  184     0316  1     ENFORCE_RECORD,                     ! Set to TRUE if the current language
;  185     0317  1                                         !      requires that COMP be a
;  186     0318  1                                         !      component of REC in the
;  187     0319  1                                         !      expression REC.COMP
;  188     0320  1     EXPRESSION_RADIX,                   ! Radix value to be used to interpret
;  189     0321  1                                         !      constants in expressions
;  190     0322  1     IDENT_OPERATOR_TABLE:               ! Pointer to an Operator Table for the
;  191     0323  1                 REF VECTOR[,LONG],      !      current language where the sym-
;  192     0324  1                                         !      bols are identifiers (NOT, AND)
;  193     0325  1     INCOMPLETE_QUAL,                    ! Set to TRUE if incomplete data
;  194     0326  1                                         !      qualification is allowed in
;  195     0327  1                                         !      then current language.
;  196     0328  1                                         !      (e.g., A.C in place of A.B.C)
;  197     0329  1     MULTIPLE_SUBSCR,                    ! Set to TRUE if multiple subscripts
;  198     0330  1                                         !      parens (X[1,2][3]) are allowed
;  199     0331  1     OPCHAR_OPERATOR_TABLE:              ! Pointer to an Operator Table for the
;  200     0332  1                 REF VECTOR[,LONG],      !      current language where the sym-
;  201     0333  1                                         !      bols are special characters (+,**)
;  202     0334  1     PRIMARY_TABLE: REF PRIMARY$TABLE,   ! Pointer to the Primary Parser State
;  203     0335  1                                         !      Table for the current language
```

```
;  204     0336  1     SAVED_TOKEN: INITIAL(0),                 ! Token pointer saved by Primary Parser
;  205     0337  1                                              !     until next call by Expr. Parser
;  206     0338  1     STATE_TABLE: REF NUMST$TABLE,            ! Pointer to the Number Scanner State
;  207     0339  1                                              !     Table for the current language
;  208     0340  1     SUBSCRIPT_TERM_TBL:                      ! Pointer to Terminator Table to use
;  209     0341  1                 REF VECTOR[,LONG],           !     when picking up subscripts
;  210     0342  1     PRIDTBL: REF VECTOR[,LONG],              ! Pointer to Predefined Identifier Table
;  211     0343  1     TERMINATOR_CODE:                         ! Terminator code for last terminator
;  212     0344  1         INITIAL(TOKEN$K_TERM_NONE),          !     token found by lexical scanner
;  213     0345  1     TERMINATOR_LENGTH: INITIAL(0),           ! Length in characters of last termina-
;  214     0346  1                                              !     tor found by the lexical scanner
;  215     0347  1     VARSTACK1: VECTOR[VARSTK_SIZE],          ! Stack of Variant Set RST Entry pointers
;  216     0348  1     VARSTACK2: VECTOR[VARSTK_SIZE],          ! Stack of RST Variant Entry pointers
;  217     0349  1     VARSTACK3: VECTOR[VARSTK_SIZE],          ! Stack of variant component indicies
;  218     0350  1     VARSTK_INDEX;                            ! Current index for VARSTACKn stacks
```

```
;  220     0351  1 !       M A C R O S   T O   G E N E R A T E   P A R S E   T A B L E S
;  221     0352  1 !
;  222     0353  1 !
;  223     0354  1 !
;  224     0355  1 !       These macros are used to generate the parse tables used by the Lexical
;  225     0356  1 !       Scanner and the Parser to parse both language-independent and language-
;  226     0357  1 !       specific constructs accepted by DEBUG.
;  227     0358  1 !
;  228     0359  1 !
;  229     0360  1 !
;  230     0361  1 ! CHARACTER TABLE
;  231     0362  1 !
;  232     0363  1 !
;  233     0364  1 ! Define the macros which generate the Character table.  This includes the
;  234     0365  1 ! macros which generate the base Character Table for language UNKNOWN and
;  235     0366  1 ! the macros which generate the Character Exception Tables for the individual
;  236     0367  1 ! languages.  The Character Exception Table for a language lists those char-
;  237     0368  1 ! acters which have different characteristics in that language that the de-
;  238     0369  1 ! fault characteristics specified for language UNKNOWN.  The Character Table
;  239     0370  1 ! for a language is thus formed by copying the default table for language
;  240     0371  1 ! UNKNOWN and then overlaying the entries for those characters listed in the
;  241     0372  1 ! language's Character Exception Table.
;  242     0373  1 !
;  243     0374  1 ! These macros are used as follows.  To generate the default Character Table,
;  244     0375  1 ! the following sequence of macro invocations is used:
;  245     0376  1 !
;  246     0377  1 !       CHAR_TABLE(TBLNAME,
;  247     0378  1 !               CHAR_ENT(CHAR, CLASS, BIT1, BIT2, ...)
;  248     0379  1 !               ...
;  249     0380  1 !               CHAR_ENT(CHAR, CLASS, BIT1, BIT2, ...));
;  250     0381  1 !
;  251     0382  1 ! Here TBLNAME is the name of the table to be built.  The CHAR_TABLE macro
;  252     0383  1 ! defines the whole Character Table as a BLOCKVECTOR and causes each element
;  253     0384  1 ! in the vector to be filled with zeroes unless this is overridden by an
;  254     0385  1 ! explicit CHAR_ENT invocation for that specific character.
;  255     0386  1 !
;  256     0387  1 ! The CHAR_ENT macro sets the character characteristics for a specified
;  257     0388  1 ! character.  Here CHAR is the character itself (e.g., 'A'), CLASS is the
;  258     0389  1 ! Number Scanner Character Class, and BIT1, BIT2, ... are the names of the
;  259     0390  1 ! other characteristics bits to be set for this character.  The CLASS param-
;  260     0391  1 ! eter is automatically prefixed by "NUMST$K_CLASS_" by the macro.  Also,
;  261     0392  1 ! the BITn parameters are automatically prefixed by "CHRTBL$M_" to generate
;  262     0393  1 ! the proper mask value names.
;  263     0394  1 !
;  264     0395  1 ! To generate a Character Exception Table for a language, the following
;  265     0396  1 ! macro invocations are used:
;  266     0397  1 !
;  267     0398  1 !       CHAR_EXCEPTION_TABLE(TBLNAME,
;  268     0399  1 !               CHAR_ENTRY(CHAR, CLASS, BIT1, BIT2, ...)
;  269     0400  1 !               ...
;  270     0401  1 !               CHAR_ENTRY(CHAR, CLASS, BIT1, BIT2, ...));
;  271     0402  1 !
;  272     0403  1 ! Here CHAR, CLASS, and BIT1, BIT2, ... have the same meanings as for the
;  273     0404  1 ! CHAR_ENT macro above.  Similarly, TBLNAME is the name given to the new
;  274     0405  1 ! Character Exception Table.  The only difference between these macros and
;  275     0406  1 ! those described above is that a different data structure is generated.
;  276     0407  1 !
```

```
;  277          0408  1 MACRO
;  278        M 0409  1     CHAR_TABLE(TBLNAME) =
;  279          0410  1         OWN TBLNAME: VECTOR[256,LONG] PSECT(DBG$PLIT) PRESET(%REMAINING) %;
;  280          0411  1
;  281          0412  1 MACRO
;  282        M 0413  1     CHAR_ENT(CHAR, CLASS) =
;  283          0414  1         [CHAR]=((%NAME('NUMST$K_CLASS_',CLASS)^4) OR CHAR_FLAGS(%REMAINING)) %;
;  284          0415  1
;  285          0416  1 MACRO
;  286        M 0417  1     CHAR_FLAGS[FLAG] =
;  287          0418  1         %NAME('CHRTBL$M_',FLAG) %;
;  288          0419  1
;  289          0420  1 MACRO
;  290        M 0421  1     CHAR_EXCEPTION_TABLE(TBLNAME) =
;  291        M 0422  1         %IF %LENGTH EQL 1
;  292        M 0423  1         %THEN
;  293        M 0424  1             BIND TBLNAME = PLIT(REP 0 OF (0)): VECTOR[,LONG]
;  294        M 0425  1         %ELSE
;  295        M 0426  1             BIND TBLNAME = PLIT(%REMAINING): VECTOR[,LONG]
;  296          0427  1         %FI %;
;  297          0428  1
;  298          0429  1 MACRO
;  299        M 0430  1     CHAR_ENTRY(CHAR, CLASS) =
;  300          0431  1         CHAR^24 OR %NAME('NUMST$K_CLASS_',CLASS)^4 OR CHAR_FLAGS(%REMAINING) %;
;  301          0432  1
;  302          0433  1
;  303          0434  1 ! Define fields of Character Exception Table entry.  These definitions are only
;  304          0435  1 ! used in the DBG$PARSER_SET_LANGUAGE routine below.
;  305          0436  1 !
;  306          0437  1 FIELD CE_FLDS =
;  307          0438  1         SET
;  308          0439  1         CE_BITS = [ 0, V (0,24) ],      ! Character characteristics bits
;  309          0440  1         CE_CHAR = [ 0, B3_ ]            ! The character itself
;  310          0441  1         TES;
;  311          0442  1
;  312          0443  1 MACRO
;  313          0444  1     CE_ENTRY = BLOCK[1,WORD] FIELD(CE_FLDS) %;
;  314          0445  1
;  315          0446  1
;  316          0447  1
;  317          0448  1 ! OPERATOR TABLE
;  318          0449  1 !
;  319          0450  1 !
;  320          0451  1 ! Define the macros which generate an Operator Table for a language.  An Opera-
;  321          0452  1 ! tor Table is a counted vector (a PLIT) of longwords which point to Operator
;  322          0453  1 ! Lexical Token Entries for the operators of the language.  Each pointer is
;  323          0454  1 ! relative to the address TABLEBASE so that the code is completely Position-
;  324          0455  1 ! Independent (PIC); the true pointer value is thus the longword value plus
;  325          0456  1 ! TABLEBASE.
;  326          0457  1 !
;  327          0458  1 ! An Operator Table is generated with the following macro invocations:
;  328          0459  1 !
;  329          0460  1 !    OPERATOR_TABLE(TBLNAME,
;  330          0461  1 !       OPERATOR_ENTRY(OPNAME,CODE,KIND,LEFT_PREC,RIGHT_PREC,FLAG1,...),
;  331          0462  1 !                 .
;  332          0463  1 !       OPERATOR_ENTRY(OPNAME,CODE,KIND,LEFT_PREC,RIGHT_PREC,FLAG1,...));
;  333          0464  1 !
```

```
;  334          0465  1 ! Here TBLNAME is the name of the Operator Table generated.  OPNAME is the
;  335          0466  1 ! quoted character string which constitutes the operator (e.g., '+', ':=',
;  336          0467  1 ! 'AND', '.EQ.'), CODE is the name of the operation to be performed (e.g.,
;  337          0468  1 ! SUBSCRIPT, ADD, OPENPAREN--this is automatically prefixed by "TOKEN$K_"
;  338          0469  1 ! by the macro), KIND is the operator kind (PREFIX, INFIX, or POSTFIX),
;  339          0470  1 ! LEFT_PREC is its left precedence, and RIGHT_PREC is its right precedence.
;  340          0471  1 ! FLAGT, ... is zero or more optional parameters which specify flag bits
;  341          0472  1 ! to be set in the Token Entry.  Each FLAGn name is automatically prefixed
;  342          0473  1 ! by "TOKEN$M_" by the macro to generate the appropriate mask value.  The
;  343          0474  1 ! only flag at present is PRIMARY which means that the operator is an
;  344          0475  1 ! operator within a Primary Symbol.
;  345          0476  1 !
;  346          0477  1 ! The OPERATOR_ENTRY macro can also be used independently to generate an
;  347          0478  1 ! Operator Lexical Token Entry.  It returns the address of the Operator
;  348          0479  1 ! Lexical Token Entry it created.
;  349          0480  1 !
;  350          0481  1 MACRO
;  351        M 0482  1     OPERATOR_TABLE(TBLNAME) =
;  352        M 0483  1         %IF %LENGTH EQL 1
;  353        M 0484  1         %THEN
;  354        M 0485  1             BIND TBLNAME = PLIT(REP 0 OF (0)): VECTOR[,LONG]
;  355        M 0486  1         %ELSE
;  356        M 0487  1             BIND TBLNAME = PLIT(OP_ENT(%REMAINING)): VECTOR[,LONG]
;  357          0488  1         %FI %;
;  358          0489  1
;  359          0490  1 MACRO
;  360        M 0491  1     OP_ENT[ADDRESS] =
;  361          0492  1         (ADDRESS) - TABLEBASE %;
;  362          0493  1
;  363          0494  1 MACRO
;  364        M 0495  1     OPERATOR_ENTRY(OPNAME,CODE,KIND,LEFT_PREC,RIGHT_PREC) =
;  365        M 0496  1         UPLIT BYTE(%NAME('TOKEN$K_',KIND,'_OP'),
;  366        M 0497  1                 %IF %LENGTH LEQ 5
;  367        M 0498  1                 %THEN
;  368        M 0499  1                     0
;  369        M 0500  1                 %ELSE
;  370        M 0501  1                     0 OR OP_FLAGS(%REMAINING)
;  371        M 0502  1                 %FI,
;  372        M 0503  1                 WORD (%NAME('TOKEN$K_',CODE)),
;  373        M 0504  1                 RIGHT_PREC, LEFT_PREC, 0, 0, 0, 0, 0 , 0,
;  374          0505  1                 %ASCIC OPNAME) %;
;  375          0506  1
;  376          0507  1 MACRO
;  377        M 0508  1     OP_FLAGS[FLAGNAME] =
;  378          0509  1         %NAME('TOKEN$M_',FLAGNAME) %;
;  379          0510  1
;  380          0511  1
;  381          0512  1
;  382          0513  1 ! BUILT-IN-FUNCTION TABLE
;  383          0514  1 !
;  384          0515  1 !
;  385          0516  1 ! Define the macros which generate an Built-in Function Table for a language.
;  386          0517  1 ! A Built-in Function Table is a counted vector (a PLIT) of longwords which
;  387          0518  1 ! point to Operand Lexical Token Entries for the Built-in Functions of the
;  388          0519  1 ! language.  Each pointer is relative to the address TABLEBASE so that the
;  389          0520  1 ! code is completely Position-Independent (PIC); the true pointer value is
;  390          0521  1 ! thus the longword value plus TABLEBASE.
```

```
;  391          0522  1 !
;  392          0523  1 ! An Built-in Function Table is generated with the following macro invocations:
;  393          0524  1 !
;  394          0525  1 !    BUILT_IN_FUNCTION_TABLE(TBLNAME,
;  395          0526  1 !       BUILT_IN_FUNCTION_ENTRY(OPNAME,CODE),
;  396          0527  1 !                ...
;  397          0528  1 !       BUILT_IN_FUNCTION_ENTRY(OPNAME,CODE);
;  398          0529  1 !
;  399          0530  1 ! Here TBLNAME is the name of the Built-in Function Table generated.  OPNAME
;  400          0531  1 ! is the quoted character string which constitutes the Built-in Function call
;  401          0532  1 ! (e.g., 'succ', 'pred'), CODE is the name of the Built-in Function to be
;  402          0533  1 ! performed (e.g, SUCC, PRED--this is automatically prefixed by "TOKEN$K_"
;  403          0534  1 ! by the macro), and ARGUMENTS is the number of arguments for the Built-in
;  404          0535  1 ! function.
;  405          0536  1 !
;  406          0537  1 ! The BUILT_IN_FUNCTION_ENTRY macro can also be used independently to
;  407          0538  1 ! generate an Built-in Function Lexical Token Entry.  It returns the
;  408          0539  1 ! address of the Built-in Function Lexical Token Entry it created.
;  409          0540  1 !
;  410          0541  1 MACRO
;  411        M 0542  1     BUILT_IN_FUNCTION_TABLE(TBLNAME) =
;  412        M 0543  1         %IF %LENGTH EQL 1
;  413        M 0544  1         %THEN
;  414        M 0545  1             BIND TBLNAME = PLIT(REP 0 OF (0)): VECTOR[,LONG]
;  415        M 0546  1         %ELSE
;  416        M 0547  1             BIND TBLNAME = PLIT(BIF_ENT(%REMAINING)): VECTOR[,LONG]
;  417          0548  1         %FI %;
;  418          0549  1
;  419          0550  1 MACRO
;  420        M 0551  1     BIF_ENT[ADDRESS] =
;  421          0552  1         (ADDRESS) - TABLEBASE %;
;  422          0553  1
;  423          0554  1 MACRO
;  424        M 0555  1     BUILT_IN_FUNCTION_ENTRY(OPNAME,CODE,ARGUMENTS) =
;  425        M 0556  1         UPLIT BYTE(TOKEN$K_OPERAND, ARGUMENTS,
;  426        M 0557  1                 WORD(TOKEN$K_BUILTIN_FUNCTION),
;  427        M 0558  1                 BYTE(%NAME('TOKEN$K_',CODE)), 0,
;  428        M 0559  1                 0, 0,
;  429          0560  1                 %ASCIC OPNAME) %;
;  430          0561  1
;  431          0562  1
;  432          0563  1 ! Define two tokens that are used in other modules.
;  433          0564  1 !
;  434          0565  1 GLOBAL BIND
;  435          0566  1     DBG$GL_CONVERT_TOKEN =
;  436          0567  1         OPERATOR_ENTRY ('CONVERT', CONVERT, INFIX, 0, 0),
;  437          0568  1     DBG$GL_DEPOSIT_TOKEN =
;  438          0569  1         OPERATOR_ENTRY ('DEPOSIT', DEPOSIT, INFIX, 0, 0),
;  439          0570  1     DBG$GL_IDENTITY_TOKEN =
;  440          0571  1         OPERATOR_ENTRY ('IDENTITY', IDENTITY, PREFIX, 0, 0),
;  441          0572  1     DBG$GL_NEG_CONST_TOKEN =
;  442          0573  1         OPERATOR_ENTRY ('NEGCONST', NEGCONST, PREFIX, 0, 0),
;  443          0574  1     DBG$GL_POS_CONST_TOKEN =
;  444          0575  1         OPERATOR_ENTRY ('POSCONST', POSCONST, PREFIX, 0, 0),
;  445          0576  1     DBG$GL_NEG_SIGN_TOKEN =
;  446          0577  1         OPERATOR_ENTRY ('-', UNARY_MINUS, PREFIX, 0, 0),
;  447          0578  1     DBG$GL_POS_SIGN_TOKEN =
```

```
;  448          0579  1          OPERATOR_ENTRY ('+', UNARY_PLUS, PREFIX, 0, 0);
;  449          0580  1
;  450          0581  1
;  451          0582  1  ! OPERAND LEXICAL TOKEN ENTRIES
;  452          0583  1  !
;  453          0584  1  !
;  454          0585  1  ! Define macro which builds an Operand Lexical Token Entry and returns the
;  455          0586  1  ! address of the entry.  The macro is used like this:
;  456          0587  1  !
;  457          0588  1  !                 OPERAND_ENTRY(TOKENCODE, NAMESTRING)
;  458          0589  1  !
;  459          0590  1  ! Here TOKENCODE is the code value for the type of operand this token
;  460          0591  1  ! constitutes.  TOKEN$K_IDENTIFIER or TOKEN$K_INTEGER are valid examples.
;  461          0592  1  ! NAMESTRING is the ASCII name to be associated with the operand token.
;  462          0593  1  !
;  463          0594  1  MACRO
;  464        M 0595  1      OPERAND_ENTRY(CODE, NAMESTRING) =
;  465        M 0596  1          UPLIT BYTE(TOKEN$K_OPERAND, 0, WORD (CODE),
;  466        M 0597  1                             0, 0, 0, 0,
;  467          0598  1                             %ASCIC NAMESTRING): TOKEN$ENTRY %;
;  468          0599  1
;  469          0600  1
;  470          0601  1
;  471          0602  1  ! PREDEFINED IDENTIFIER TABLE
;  472          0603  1  !
;  473          0604  1  !
;  474          0605  1  ! Define the macros which generate a Predefined Identifier Table for a language.
;  475          0606  1  ! A Predefined Identifier Table is a counted vector (a PLIT) of longwords which
;  476          0607  1  ! point to Predefined Identifier Entries for the reserved names of the language.
;  477          0608  1  ! Each pointer is relative to the address TABLEBASE so that the code is completely
;  478          0609  1  ! Position Independent (PIC); the true pointer value is thus the longword value
;  479          0610  1  ! plus TABLEBASE.
;  480          0611  1  !
;  481          0612  1  ! A Predefined Identifier Table is generated with the following macro invocations:
;  482          0613  1  !
;  483          0614  1  !    PRID_TABLE(TBLNAME,
;  484          0615  1  !       PRID_ENTRY(PRIDNAME, FCODE, DTYPE, VALUE),
;  485          0616  1  !                  .
;  486          0617  1  !       PRID_ENTRY(PRIDNAME, FCODE, DTYPE, VALUE));
;  487          0618  1  !
;  488          0619  1  ! Here TBLNAME is the name of the Predefined ID Table generated.  PRIDNAME is
;  489          0620  1  ! the quoted character string which constitutes the reserved name (e.g.,
;  490          0621  1  ! 'TRUE', 'FALSE', '.TRUE.', '.FALSE.'), DTYPE is the Data type (e.g., Boolean,
;  491          0622  1  ! Integer).  DTYPE is automatically prefixed by 'DSC$K_DTYPE_', and VALUE is a
;  492          0623  1  ! longword constant value for the PRID constant, FCODE is the Format Code.
;  493          0624  1  !
;  494          0625  1  ! The PRID_ENTRY macro can also be used independently to generate an entry.
;  495          0626  1  ! It returns the address of the PRID Entry it created.
;  496          0627  1  !
;  497          0628  1  MACRO
;  498        M 0629  1      PRID_TABLE(TBLNAME) =
;  499        M 0630  1          %IF %LENGTH EQL 1
;  500        M 0631  1          %THEN
;  501        M 0632  1              BIND TBLNAME = PLIT(REP 0 OF (0)): VECTOR[,LONG]
;  502        M 0633  1          %ELSE
;  503        M 0634  1              BIND TBLNAME = PLIT(PRID_ENT(%REMAINING)): VECTOR[,LONG]
;  504          0635  1          %FI %;
```

```
;  505          0636  1
;  506          0637  1 MACRO
;  507        M 0638  1     PRID_ENT[ADDRESS] =
;  508          0639  1         (ADDRESS) - TABLEBASE %;
;  509          0640  1
;  510          0641  1 MACRO
;  511        M 0642  1     PRID_ENTRY(NAMESTRING, FCODE, DTYPE, VALUE) =
;  512        M 0643  1         UPLIT BYTE(PRID$K_CONSTANT,
;  513        M 0644  1                    %NAME('DSC$K_DTYPE_', DTYPE),
;  514        M 0645  1                    %NAME('RST$K_TYPE_', FCODE),
;  515        M 0646  1                    0,
;  516        M 0647  1                    LONG (VALUE),
;  517        M 0648  1                    %ASCIC NAMESTRING)%;
;  518          0649  1
;  519          0650  1
;  520          0651  1
;  521          0652  1 ! TERMINATOR LEXICAL TOKEN ENTRIES
;  522          0653  1 !
;  523          0654  1 !
;  524          0655  1 ! Define the macros which generate Terminator Lexical Token Entries.  These
;  525          0656  1 ! entries define lexical tokens which can terminate the current expression.
;  526          0657  1 ! For example, subscript expressions can normally be terminated by the tokens
;  527          0658  1 ! "," and ")".  These are thus the Terminator Tokens for subscript expressions.
;  528          0659  1 ! Similarly, "DO" is a terminator token for the address expression in the
;  529          0660  1 ! SET BREAK command and "=" is a terminator token for the address expression
;  530          0661  1 ! in the DEPOSIT command.  Terminator Lexical Token Entries thus define the
;  531          0662  1 ! tokens which can validly terminate the current expression in the current
;  532          0663  1 ! context.  Terminator Lexical Token Entries have exactly the same format as
;  533          0664  1 ! Operand Lexical Token Entries are are referenced with the same field names.
;  534          0665  1 !
;  535          0666  1 ! A table of Terminator Lexical Token Entries is declared as follows:
;  536          0667  1 !
;  537          0668  1 !       TERMINATOR_TABLE(TBLNAME,
;  538          0669  1 !               TERMINATOR_ENTRY(NAMESTRING, CODE, FLAG1, ...),
;  539          0670  1 !               ...
;  540          0671  1 !               TERMINATOR_ENTRY(NAMESTRING, CODE, FLAG1, ...));
;  541          0672  1 !
;  542          0673  1 ! Here TBLNAME is the name of the terminator table.  NAMESTRING is the ASCII
;  543          0674  1 ! string which constitutes the terminator token, CODE is the name of a termi-
;  544          0675  1 ! nator token code which identifies what kind of terminator this is (this name
;  545          0676  1 ! is automatically prefixed by "TOKEN$K_" by the macro), and FLAG1, ... is
;  546          0677  1 ! zero or more flag names indicating flag bits to be set in the Terminator
;  547          0678  1 ! Token Entry.  The flags names are automatically prefixed by "TOKEN$M_" to
;  548          0679  1 ! generate the appropriate mask value name.
;  549          0680  1 !
;  550          0681  1 MACRO
;  551        M 0682  1     TERMINATOR_TABLE(TBLNAME) =
;  552        M 0683  1         %IF %LENGTH EQL 1
;  553        M 0684  1         %THEN
;  554        M 0685  1             BIND TBLNAME = PLIT(REP 0 OF (0)): VECTOR[,LONG]
;  555        M 0686  1         %ELSE
;  556        M 0687  1             BIND TBLNAME = PLIT(OP_ENT(%REMAINING)): VECTOR[,LONG]
;  557          0688  1         %FI %;
;  558          0689  1
;  559          0690  1 MACRO
;  560        M 0691  1     TERMINATOR_ENTRY(NAMESTRING, CODE) =
;  561        M 0692  1         UPLIT BYTE(0,
```

```
;  562          M 0693  1                %IF %LENGTH LEQ 2
;  563          M 0694  1                %THEN
;  564          M 0695  1                    0
;  565          M 0696  1                %ELSE
;  566          M 0697  1                    0 OR OP_FLAGS(%REMAINING)
;  567          M 0698  1                %FI,
;  568          M 0699  1                WORD (%NAME('TOKEN$K_',CODE)),
;  569          M 0700  1                0, 0, 0, 0,
;  570            0701  1                %ASCIC NAMESTRING) %;
;  571            0702  1
;  572            0703  1
;  573            0704  1
;  574            0705  1  ! NUMBER SCANNER STATE TABLE
;  575            0706  1  !
;  576            0707  1  !
;  577            0708  1  ! Define the macros which generate Number Scanner State Tables.  These tables
;  578            0709  1  ! are used in the lexical scanning of numeric constants and are in general
;  579            0710  1  ! specific to each language.
;  580            0711  1  !
;  581            0712  1  ! A Number Scanner State Table for a language is generated with the following
;  582            0713  1  ! macro invocations:
;  583            0714  1  !
;  584            0715  1  !       NUMBER_STATE_TABLE(TBLNAME,
;  585            0716  1  !
;  586            0717  1  !               NUMBER_STATE(STATE_ID,
;  587            0718  1  !                       NUMBER_TRANSITION(CHAR_CLASS, ACTION, NEXTSTATE),
;  588            0719  1  !                       ...
;  589            0720  1  !                       NUMBER_TRANSITION(CHAR_CLASS, ACTION, NEXTSTATE),
;  590            0721  1  !                       NUMBER_TRANSITION(OTHER, ACTION, NEXTSTATE)),
;  591            0722  1  !
;  592            0723  1  !                   ...
;  593            0724  1  !
;  594            0725  1  !               NUMBER_STATE(STATE_ID,
;  595            0726  1  !                       NUMBER_TRANSITION(CHAR_CLASS, ACTION, NEXTSTATE),
;  596            0727  1  !                       ...
;  597            0728  1  !                       NUMBER_TRANSITION(CHAR_CLASS, ACTION, NEXTSTATE),
;  598            0729  1  !                       NUMBER_TRANSITION(OTHER, ACTION, NEXTSTATE)));
;  599            0730  1  !
;  600            0731  1  ! The NUMBER_STATE_TABLE macro sets up the Number Scanner State Table as a
;  601            0732  1  ! whole and binds the table name TBLNAME to that structure.  Each state in the
;  602            0733  1  ! table is declared with the NUMBER_STATE macro whose STATE_ID argument names
;  603            0734  1  ! the state.  The actual state name is prefixed by 'NUMST$K_STATE_' by the
;  604            0735  1  ! macro and must be declared as such in the COMPILETIME declaration below.
;  605            0736  1  !
;  606            0737  1  ! The NUMBER_TRANSITION macro defines one transition from the current state to
;  607            0738  1  ! some other state.  The transition is taken if the next input character is of
;  608            0739  1  ! the character class specified by CHAR_CLASS.  (CHAR_CLASS is automatically
;  609            0740  1  ! prefixed by 'NUMST$K_CLASS_' to generate the class code constant.)  If the
;  610            0741  1  ! transition is taken, the action specified by ACTION is taken.  ACTION, which
;  611            0742  1  ! is automatically prefixed by 'NUMST$K_ACT_' by the macro, is a CASE index
;  612            0743  1  ! used by the Number Scanner to select an action routine which performs what-
;  613            0744  1  ! ever semantic processing is appropriate.  After the action routine executes,
;  614            0745  1  ! the next state in the state table is given by NEXTSTATE.  NEXTSTATE is auto-
;  615            0746  1  ! matically prefixed by 'NUMST$K_STATE_' by the macro and must be declared as
;  616            0747  1  ! the STATE_ID on some other NUMBER_STATE declaration in the state table.
;  617            0748  1  !
;  618            0749  1  ! Every state in the state table must have at least two transitions and the
```

```
;  619          0750  1 ! last transition must always be for class NUMST$K_CLASS_OTHER, written as
;  620          0751  1 ! OTHER in the NUMBER_TRANSITION invocation.  (There is no point to having
;  621          0752  1 ! a state with only a single transition since that state could then be merged
;  622          0753  1 ! with the transition's next state.)
;  623          0754  1 !
;  624          0755  1 MACRO
;  625        M 0756  1     NUMBER_STATE_TABLE(TBLNAME) =
;  626        M 0757  1         BIND TBLNAME = UPLIT(%REMAINING): NUMST$TABLE
;  627          0758  1         %ASSIGN(NUMST$XX_CUR_LOC, 0) %;
;  628          0759  1
;  629          0760  1 MACRO
;  630        M 0761  1     NUMBER_STATE(STATE_ID) =
;  631        M 0762  1         %ASSIGN(%NAME('NUMST$XX_STATE_', STATE_ID), NUMST$XX_CUR_LOC)
;  632        M 0763  1         %ASSIGN(NUMST$XX_CUR_LOC, NUMST$XX_CUR_LOC + (%LENGTH - 1)/3)
;  633          0764  1         %REMAINING %;
;  634          0765  1
;  635          0766  1 MACRO
;  636        M 0767  1     NUMBER_TRANSITION(CHAR_CLASS, ACTION, NEXTSTATE) =
;  637        M 0768  1         BYTE (%NAME('NUMST$K_CLASS_', CHAR_CLASS)),
;  638        M 0769  1         BYTE (%NAME('NUMST$K_ACT_', ACTION)),
;  639          0770  1         WORD (%NAME('NUMST$XX_STATE_', NEXTSTATE)) %;
;  640          0771  1
;  641          0772  1 COMPILETIME
;  642          0773  1     NUMST$XX_CUR_LOC = 0,                   ! Current index into state table vector
;  643          0774  1                                             !     during macro expansion
;  644          0775  1     NUMST$XX_STATE_START_STATE = 0,         ! Index values in state table vector
;  645          0776  1     NUMST$XX_STATE_LEADING_DOT = 0,         !     for state table entries
;  646          0777  1     NUMST$XX_STATE_ACCUM_INT = 0,
;  647          0778  1     NUMST$XX_STATE_T_ACCUM_INT = 0,
;  648          0779  1     NUMST$XX_STATE_ACCUM_HEX = 0,
;  649          0780  1     NUMST$XX_STATE_ACCUM_FRAC = 0,
;  650          0781  1     NUMST$XX_STATE_T_ACCUM_FRAC = 0,
;  651          0782  1     NUMST$XX_STATE_GET_EXPONENT = 0,
;  652          0783  1     NUMST$XX_STATE_GET_EXP_SIGN = 0,
;  653          0784  1     NUMST$XX_STATE_ACCUM_EXP = 0,
;  654          0785  1     NUMST$XX_STATE_T_ACCUM_EXP = 0,
;  655          0786  1     NUMST$XX_STATE_B_START_STATE = 0,
;  656          0787  1     NUMST$XX_STATE_B_ACCUM_INT = 0,
;  657          0788  1     NUMST$XX_STATE_T_B_ACCUM_INT = 0,
;  658          0789  1     NUMST$XX_STATE_B_ACCUM_FRAC = 0,
;  659          0790  1     NUMST$XX_STATE_T_B_ACCUM_FRAC = 0,
;  660          0791  1     NUMST$XX_STATE_END_STATE = 0;
;  661          0792  1
;  662          0793  1
;  663          0794  1 ! PRIMARY PARSER STATE TABLE
;  664          0795  1 !
;  665          0796  1 !
;  666          0797  1 ! Define the macros which generate Primary Parser State Tables.  These tables
;  667          0798  1 ! are used in the parsing of Primary Symbols (symbol names including pathname
;  668          0799  1 ! qualification, data qualification, subscripting, dereferencing, etc.) and
;  669          0800  1 ! are specific to each language.
;  670          0801  1 !
;  671          0802  1 ! A Primary Parser State Table for a language is generated with the following
;  672          0803  1 ! macro invocations:
;  673          0804  1 !
;  674          0805  1 !       PRIMARY_STATE_TABLE(TBLNAME,
;  675          0806  1 !
```

```
;  676          0807  1 !           PRIMARY_STATE(STATE_ID,
;  677          0808  1 !               PRIMARY_TRANSITION(OPCODE, ACTION, NEXTSTATE),
;  678          0809  1 !                   ...
;  679          0810  1 !               PRIMARY_TRANSITION(OPCODE, ACTION, NEXTSTATE)),
;  680          0811  1 !
;  681          0812  1 !           ...
;  682          0813  1 !
;  683          0814  1 !           PRIMARY_STATE(STATE_ID,
;  684          0815  1 !               PRIMARY_TRANSITION(OPCODE, ACTION, NEXTSTATE),
;  685          0816  1 !                   ...
;  686          0817  1 !               PRIMARY_TRANSITION(OPCODE, ACTION, NEXTSTATE)));
;  687          0818  1 !
;  688          0819  1 ! The PRIMARY_STATE_TABLE macro sets up the Primary Parser State Table as a
;  689          0820  1 ! whole and binds the table name TBLNAME to that data structure.  Each state in
;  690          0821  1 ! the table is declared with the PRIMARY_STATE macro whose STATE_ID argument
;  691          0822  1 ! names the state.  The actual state name is prefixed by "PRIMARY$XX_STATE_"
;  692          0823  1 ! by the macro and must be declared in the COMPILETIME declaration below.
;  693          0824  1 !
;  694          0825  1 ! The PRIMARY_TRANSITION macro defines one transition from the current state
;  695          0826  1 ! to some other state.  The transition is taken if the current Primary Operator
;  696          0827  1 ! returned by the Lexical Scanner has the Operator Code (TOKEN$W_CODE) speci-
;  697          0828  1 ! fied as OPCODE.  (OPCODE is automatically prefixed by "TOKEN$K_" to generate
;  698          0829  1 ! the constant name.)  If the transition is taken, the action routine specified
;  699          0830  1 ! by ACTION is taken.  ACTION, which the macro prefixes with "PRIMARY$K_ACT_",
;  700          0831  1 ! is a CASE index used by the Primary Parser to select an action routine to do
;  701          0832  1 ! whatever semantic processing is appropriate to build the Primary Descriptor
;  702          0833  1 ! for the Primary Symbol being parsed.  After the action routine executes, the
;  703          0834  1 ! next state in the state table is given by NEXTSTATE.  NEXTSTATE is automati-
;  704          0835  1 ! cally prefixed by "PRIMARY$XX_STATE_" by the macro and must be declared as
;  705          0836  1 ! the STATE_ID on some other PRIMARY_STATE declaration in the state table.
;  706          0837  1 !
;  707          0838  1 ! If a Primary Operator is encountered during the scan for which there is no
;  708          0839  1 ! transition in the current Primary Parser State Table state, the Primary
;  709          0840  1 ! Parser signals an error.  This is how many ill-formed Primary Symbols are
;  710          0841  1 ! detected.  Every state thus has an implicit transition for every unspecified
;  711          0842  1 ! Operator Code (the "other" transition) for which the action is to signal a
;  712          0843  1 ! syntax error.
;  713          0844  1 !
;  714          0845  1 MACRO
;  715        M 0846  1     PRIMARY_STATE_TABLE(TBLNAME) =
;  716        M 0847  1         BIND TBLNAME = UPLIT(%REMAINING): PRIMARY$TABLE
;  717          0848  1         %ASSIGN(PRIMARY$XX_CUR_LOC, 0) %;
;  718          0849  1
;  719          0850  1 MACRO
;  720        M 0851  1     PRIMARY_STATE(STATE_ID) =
;  721        M 0852  1         %ASSIGN(%NAME('PRIMARY$XX_STATE_', STATE_ID), PRIMARY$XX_CUR_LOC)
;  722        M 0853  1         %ASSIGN(PRIMARY$XX_CUR_LOC, PRIMARY$XX_CUR_LOC + (%LENGTH - 1)/3 + 1)
;  723          0854  1         %IF %LENGTH GTR 1 %THEN %REMAINING, %FI LONG (0) %;
;  724          0855  1
;  725          0856  1 MACRO
;  726        M 0857  1     PRIMARY_TRANSITION(OPCODE, ACTION, NEXTSTATE) =
;  727        M 0858  1         BYTE (%NAME('TOKEN$K_', OPCODE)),
;  728        M 0859  1         BYTE (%NAME('PRIMARY$K_ACT_', ACTION)),
;  729          0860  1         WORD (%NAME('PRIMARY$XX_STATE_', NEXTSTATE)) %;
;  730          0861  1
;  731          0862  1 COMPILETIME
;  732          0863  1     PRIMARY$XX_CUR_LOC = 0,             ! Current index into primary parser state
```

```
;  733          0864  1                                          !       table during macro expansion
;  734          0865  1      PRIMARY$XX_STATE_START_STATE = 0,    ! Index values in primary parser state
;  735          0866  1      PRIMARY$XX_STATE_GET_GLOBAL = 0,     !       table vector for the states
;  736          0867  1      PRIMARY$XX_STATE_GOT_BACKSLASH = 0,
;  737          0868  1      PRIMARY$XX_STATE_GOT_DOT = 0,
;  738          0869  1      PRIMARY$XX_STATE_GOT_SUBSCRIPT = 0,
;  739          0870  1      PRIMARY$XX_STATE_GOT_SUBSCRIPT2 = 0,
;  740          0871  1      PRIMARY$XX_STATE_GOT_BRACKET = 0,
;  741          0872  1      PRIMARY$XX_STATE_GOT_DEREF = 0,
;  742          0873  1      PRIMARY$XX_STATE_GOT_DOT_SLASH = 0,
;  743          0874  1      PRIMARY$XX_STATE_END_STATE = 0;
;  744          0875  1
;  745          0876  1
;  746          0877  1  ! The following data structure is used in the SAVE_SUBSCRIPTS routine
;  747          0878  1  ! and the PATHNAME_TO_PRIMARY routine.
;  748          0879  1  !
;  749          0880  1  FIELD
;  750          0881  1      SUBSCR_DESC_FIELDS =
;  751          0882  1          SET
;  752          0883  1          SUBSCR$B_PATH_INDEX = [0,0,8,0],        ! Says which pathname element
;  753          0884  1                                                  !    the subscripts were
;  754          0885  1                                                  !    associated with
;  755          0886  1          SUBSCR$B_SUBCNT = [0,8,8,0],            ! Count of subscripts so far
;  756          0887  1          SUBSCR$V_RANGE = [0,16,1,0],            ! True if range was specified
;  757          0888  1          SUBSCR$V_ASTER = [0,17,1,0],            ! True for "*" range
;  758          0889  1          SUBSCR$V_MARKER = [0,18,1,0],           ! Marker bit (see SAVE_SUBSCRIPTS)
;  759          0890  1          SUBSCR$L_LBOUND = [1,0,32,0],           ! Lower bound of range
;  760          0891  1          SUBSCR$L_UBOUND = [2,0,32,0]            ! Upper bound of range
;  761          0892  1          TES;
;  762          0893  1  MACRO
;  763        M 0894  1      SUBSCR$DESC = BLOCKVECTOR[DBG$K_PATHNAME_SIZE,3,LONG]
;  764          0895  1                              FIELD(SUBSCR_DESC_FIELDS)%;
;  765          0896  1  LITERAL
;  766          0897  1      SUBSCR_DESC_SIZE = 3*DBG$K_PATHNAME_SIZE*%UPVAL;
;  767          0898  1
;  768          0899  1
;  769          0900  1  ! PARSER FLAGS
;  770          0901  1  !
;  771          0902  1  !
;  772          0903  1  ! Define parser flags which may have different settings for different languages.
;  773          0904  1  ! These flags are stored in the Table of Language-Specific Tables and are copied
;  774          0905  1  ! to individual OWN variables when the current language is set.  These flags
;  775          0906  1  ! control the behavior of the parser in those cases where different languages
;  776          0907  1  ! have different behaviors and the easiest way of encoding these differences
;  777          0908  1  ! is to have a flags to control the parser.  The following flags are available:
;  778          0909  1  !
;  779          0910  1  !       MULTIPLE_SUBSCR - The language allows multiple subscript parentheses
;  780          0911  1  !                         in array references.  For example, PASCAL allows
;  781          0912  1  !                         X[1,2,3,4] to be written as X[1,2][3][4].  This
;  782          0913  1  !                         flag should not be set if the second form is not
;  783          0914  1  !                         equivalent to the first form of array reference.
;  784          0915  1  !
;  785          0916  1  ! Each of these flags is set to TRUE or FALSE in an OWN variable with the same
;  786          0917  1  ! name as the flag name give here.
;  787          0918  1
;  788          0919  1
;  789          0920  1
```

```
;  790          0921  1 ! TABLE OF LANGUAGE-SPECIFIC TABLES
;  791          0922  1 !
;  792          0923  1 !
;  793          0924  1 ! Define the macro which builds the table of pointers to the language-specific
;  794          0925  1 ! parse tables.  This table is simply a vector of pointers to the Character
;  795          0926  1 ! Exception Table, the Identifier Operator table, the Operator Character Opera-
;  796          0927  1 ! tor Table, and the Number Scanner State Table for the specified language.
;  797          0928  1 ! Each pointer is relative to TABLEBASE to keep the table position-independent
;  798          0929  1 ! (PIC).  The macro is used as follows:
;  799          0930  1 !
;  800          0931  1 !       LANGUAGE_TABLES(LANGUAGE = language-name,
;  801          0932  1 !                       CHARTBL = character-exception-table,
;  802          0933  1 !                       IDENT_OPTBL = identifier-operator-table,
;  803          0934  1 !                       OPCHAR_OPTBL = operator-character-operator-table,
;  804          0935  1 !                       NUMBER_TABLE = number-scanner-state-table,
;  805          0936  1 !                       PRIMARY_TABLE = primary-parser-state-table,
;  806          0937  1 !                       SUBSCR_TERMS = subscript-terminator-table,
;  807          0938  1 !                       PRIDTBL = Predefined-identifier-table,
;  808          0939  1 !                       BIF_TABLE = built-in-function-table,
;  809          0940  1 !                       MULTIPLE_SUBSCR = true-or-false,
;  810          0941  1 !                       ENFORCE_RECORD = true-or-false,
;  811          0942  1 !                       CASING_SIGNIFICANT = true-or-false,
;  812          0943  1 !                       COMPONENTS_IN_PATHNAME = true-or-false,
;  813          0944  1 !                       INCOMPLETE_QUAL = true-or-false);
;  814          0945  1 !
;  815          0946  1 ! Notice that keyword parameters are used.  The meanings of the parameters
;  816          0947  1 ! should be self-explanatory.
;  817          0948  1 !
;  818          0949  1 KEYWORDMACRO
;  819          0950  1     LANGUAGE_TABLES(LANGUAGE=DBG$K_UNKNOWN, CHARTBL=0, IDENT_OPTBL=0,
;  820          0951  1                     OPCHAR_OPTBL=0, NUMBER_TABLE=0, PRIMARY_TABLE=0,
;  821          0952  1                     SUBSCR_TERMS=0, PRIDTBL=0, BIF_TABLE=0,
;  822          0953  1                     MULTIPLE_SUBSCR=FALSE, ENFORCE_RECORD=TRUE,
;  823          0954  1                     CASING_SIGNIFICANT=FALSE,
;  824        M 0955  1                     COMPONENTS_IN_PATHNAME=FALSE, INCOMPLETE_QUAL=FALSE) =
;  825        M 0956  1
;  826        M 0957  1         BIND %NAME(LANGUAGE, '_TABLES') =
;  827        M 0958  1                 UPLIT  (CHARTBL - TABLEBASE,
;  828        M 0959  1                         IDENT_OPTBL - TABLEBASE,
;  829        M 0960  1                         OPCHAR_OPTBL - TABLEBASE,
;  830        M 0961  1                         NUMBER_TABLE - TABLEBASE,
;  831        M 0962  1                         PRIMARY_TABLE - TABLEBASE,
;  832        M 0963  1                         SUBSCR_TERMS - TABLEBASE,
;  833        M 0964  1                         PRIDTBL - TABLEBASE,
;  834        M 0965  1                         BIF_TABLE - TABLEBASE,
;  835        M 0966  1                         MULTIPLE_SUBSCR,
;  836        M 0967  1                         ENFORCE_RECORD,
;  837        M 0968  1                         CASING_SIGNIFICANT,
;  838        M 0969  1                         COMPONENTS_IN_PATHNAME,
;  839          0970  1                         INCOMPLETE_QUAL): VECTOR [,LONG] %;
```

```
;  841          0971  1 !       L A N G U A G E - I N D E P E N D E N T   P A R S E   T A B L E S
;  842          0972  1 !
;  843          0973  1 !
;  844          0974  1 !
;  845          0975  1 !       The Parser and Lexical Scanner tables in this section are the language-
;  846          0976  1 !       independent tables used during lexical scanning and parsing.
;  847          0977  1 !
;  848          0978  1 !
;  849          0979  1 !
;  850          0980  1 ! Define a table "base address".  The TABLEBASE label defines a location in the
;  851          0981  1 ! PLIT PSECT which constitutes the base address for all pointers within the
;  852          0982  1 ! DEBUG parse tables defined in this module.  This base address is needed to
;  853          0983  1 ! make these tables position-independent (PIC) since DEBUG may be placed any-
;  854          0984  1 ! where in the virtual address space when run with a user program.
;  855          0985  1 !
;  856          0986  1 BIND
;  857          0987  1     TABLEBASE = UPLIT BYTE (%ASCII 'BASE');
;  858          0988  1
;  859          0989  1
;  860          0990  1 ! Generate the "Percent Table" to define all built-in "%"-symbols recognized
;  861          0991  1 ! by DEBUG.  This includes %LINE, %LABEL, %NAME, and all the register names.
;  862          0992  1 !
;  863          0993  1 !
;  864          0994  1 ! Define literals which identify the kind of "%"-symbol a given such symbol
;  865          0995  1 ! is.  This is used as a CASE index for further processing in the Lexical
;  866          0996  1 ! Scanner (DBG$LEXICAL_SCANNER).
;  867          0997  1 !
;  868          0998  1 LITERAL
;  869          0999  1     PERCENT_NOFIND      = 0,            ! No such "%"-symbol exists
;  870          1000  1     PERCENT_LINE        = 1,            ! %LINE symbol
;  871          1001  1     PERCENT_LABEL       = 2,            ! %LABEL symbol
;  872          1002  1     PERCENT_NAME        = 3,            ! %NAME symbol
;  873          1003  1     PERCENT_DEC         = 4,            ! %DEC decimal operator
;  874          1004  1     PERCENT_HEX         = 5,            ! %HEX hexadecimal operator
;  875          1005  1     PERCENT_OCT         = 6,            ! %OCT octal operator
;  876          1006  1     PERCENT_BIN         = 7,            ! %BIN binary operator
;  877          1007  1     PERCENT_IDENT       = 8;            ! Identifier "%"-symbols such as %R5
;  878          1008  1
;  879          1009  1
;  880          1010  1 ! Generate the actual Percent Table itself.  Note that for compatibility with
;  881          1011  1 ! past usage, we allow abbreviations for %LINE and %LABEL, but only for those
;  882          1012  1 ! two reserved names.
;  883          1013  1 !
;  884          1014  1 BIND
;  885          1015  1     PERCENT_TABLE = PLIT(
;  886          1016  1         UPLIT BYTE(PERCENT_LINE,  %ASCIC '%LINE') - TABLEBASE,
;  887          1017  1         UPLIT BYTE(PERCENT_LINE,  %ASCIC '%LIN' ) - TABLEBASE,
;  888          1018  1         UPLIT BYTE(PERCENT_LINE,  %ASCIC '%LI'  ) - TABLEBASE,
;  889          1019  1         UPLIT BYTE(PERCENT_LABEL, %ASCIC '%LABEL')- TABLEBASE,
;  890          1020  1         UPLIT BYTE(PERCENT_LABEL, %ASCIC '%LABE') - TABLEBASE,
;  891          1021  1         UPLIT BYTE(PERCENT_LABEL, %ASCIC '%LAB' ) - TABLEBASE,
;  892          1022  1         UPLIT BYTE(PERCENT_LABEL, %ASCIC '%LA'  ) - TABLEBASE,
;  893          1023  1         UPLIT BYTE(PERCENT_NAME,  %ASCIC '%NAME') - TABLEBASE,
;  894          1024  1         UPLIT BYTE(PERCENT_IDENT, %ASCIC '%R0'  ) - TABLEBASE,
;  895          1025  1         UPLIT BYTE(PERCENT_IDENT, %ASCIC '%R1'  ) - TABLEBASE,
;  896          1026  1         UPLIT BYTE(PERCENT_IDENT, %ASCIC '%R2'  ) - TABLEBASE,
;  897          1027  1         UPLIT BYTE(PERCENT_IDENT, %ASCIC '%R3'  ) - TABLEBASE,
```

```
;  898          1028  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%R4'  ) - TABLEBASE,
;  899          1029  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%R5'  ) - TABLEBASE,
;  900          1030  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%R6'  ) - TABLEBASE,
;  901          1031  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%R7'  ) - TABLEBASE,
;  902          1032  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%R8'  ) - TABLEBASE,
;  903          1033  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%R9'  ) - TABLEBASE,
;  904          1034  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%R10' ) - TABLEBASE,
;  905          1035  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%R11' ) - TABLEBASE,
;  906          1036  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%R12' ) - TABLEBASE,
;  907          1037  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%R13' ) - TABLEBASE,
;  908          1038  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%R14' ) - TABLEBASE,
;  909          1039  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%R15' ) - TABLEBASE,
;  910          1040  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%AP'  ) - TABLEBASE,
;  911          1041  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%FP'  ) - TABLEBASE,
;  912          1042  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%SP'  ) - TABLEBASE,
;  913          1043  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%PC'  ) - TABLEBASE,
;  914          1044  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%PSL' ) - TABLEBASE,
;  915          1045  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%r0'  ) - TABLEBASE,
;  916          1046  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%r1'  ) - TABLEBASE,
;  917          1047  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%r2'  ) - TABLEBASE,
;  918          1048  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%r3'  ) - TABLEBASE,
;  919          1049  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%r4'  ) - TABLEBASE,
;  920          1050  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%r5'  ) - TABLEBASE,
;  921          1051  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%r6'  ) - TABLEBASE,
;  922          1052  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%r7'  ) - TABLEBASE,
;  923          1053  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%r8'  ) - TABLEBASE,
;  924          1054  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%r9'  ) - TABLEBASE,
;  925          1055  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%r10' ) - TABLEBASE,
;  926          1056  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%r11' ) - TABLEBASE,
;  927          1057  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%r12' ) - TABLEBASE,
;  928          1058  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%r13' ) - TABLEBASE,
;  929          1059  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%r14' ) - TABLEBASE,
;  930          1060  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%r15' ) - TABLEBASE,
;  931          1061  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%ap'  ) - TABLEBASE,
;  932          1062  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%fp'  ) - TABLEBASE,
;  933          1063  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%sp'  ) - TABLEBASE,
;  934          1064  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%pc'  ) - TABLEBASE,
;  935          1065  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%psl' ) - TABLEBASE,
;  936          1066  1          UPLIT BYTE(PERCENT_DEC,   %ASCIC '%DEC' ) - TABLEBASE,
;  937          1067  1          UPLIT BYTE(PERCENT_HEX,   %ASCIC '%HEX' ) - TABLEBASE,
;  938          1068  1          UPLIT BYTE(PERCENT_OCT,   %ASCIC '%OCT' ) - TABLEBASE,
;  939          1069  1          UPLIT BYTE(PERCENT_BIN,   %ASCIC '%BIN' ) - TABLEBASE,
;  940          1070  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%CURLOC')-TABLEBASE,
;  941          1071  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%CURVAL')-TABLEBASE,
;  942          1072  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%PREVLOC')-TABLEBASE,
;  943          1073  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%NEXTLOC')-TABLEBASE,
;  944          1074  1          UPLIT BYTE(PERCENT_IDENT, %ASCIC '%PARCNT')-TABLEBASE
;  945          1075  1                                              ): VECTOR[,LONG];
;  946          1076  1
;  947          1077  1
;  948          1078  1  ! Define Operator Lexical Token Entries for two predefined operators, namely
;  949          1079  1  ! the Initiator and the Terminator operators.  These operators do not appear
;  950          1080  1  ! explicitly in the DEBUG command line, but are used internally by the Opera-
;  951          1081  1  ! tor Precedence Parsers as expression initiator and terminator tokens.  The
;  952          1082  1  ! Primary Terminator is a pseudo-operator used to terminate Primary Symbols.
;  953          1083  1  !
;  954          1084  1  BIND
```

```
;  955          1085  1      INITIATOR_TOKEN  =
;  956          1086  1          OPERATOR_ENTRY('start of expression', INITIATOR, PREFIX, 200, 1),
;  957          1087  1      TERMINATOR_TOKEN =
;  958          1088  1          OPERATOR_ENTRY('end of expression', TERMINATOR, POSTFIX, 2, 200, LEXICAL),
;  959          1089  1      PRIMARY_TERM_TOKEN =
;  960          1090  1          OPERATOR_ENTRY('end of symbol', PRIMARY_TERM, POSTFIX, 0, 0, PRIMARY);
;  961          1091  1
;  962          1092  1
;  963          1093  1  ! Generate Operator Lexical Token Entries for the radix operators.  These
;  964          1094  1  ! are the operators which change the current expression radix back and forth.
;  965          1095  1  !
;  966          1096  1  BIND
;  967          1097  1      RADIX_OP_DEC =
;  968          1098  1          OPERATOR_ENTRY('%DEC', RADIX_DEC, PREFIX, 200, 190, LEXICAL),
;  969          1099  1      RADIX_OP_HEX =
;  970          1100  1          OPERATOR_ENTRY('%HEX', RADIX_HEX, PREFIX, 200, 190, LEXICAL),
;  971          1101  1      RADIX_OP_OCT =
;  972          1102  1          OPERATOR_ENTRY('%OCT', RADIX_OCT, PREFIX, 200, 190, LEXICAL),
;  973          1103  1      RADIX_OP_BIN =
;  974          1104  1          OPERATOR_ENTRY('%BIN', RADIX_BIN, PREFIX, 200, 190, LEXICAL);
;  975          1105  1
;  976          1106  1
;  977          1107  1  ! Generate Operand Lexical Token Entries for the built-in DEBUG symbols
;  978          1108  1  ! "." , "\", and "^" meaning current location, current value, and previous
;  979          1109  1  ! location.
;  980          1110  1  !
;  981          1111  1  BIND
;  982          1112  1      CURLOC_TOKEN  = OPERAND_ENTRY(TOKEN$K_IDENTIFIER, '%CURLOC'),
;  983          1113  1      CURVAL_TOKEN  = OPERAND_ENTRY(TOKEN$K_IDENTIFIER, '%CURVAL'),
;  984          1114  1      PREVLOC_TOKEN = OPERAND_ENTRY(TOKEN$K_IDENTIFIER, '%PREVLOC');
;  985          1115  1
;  986          1116  1
;  987          1117  1  ! Generate the Operator Table for the built-in operators allowed in Address
;  988          1118  1  ! Expressions.
;  989          1119  1  !
;  990        P 1120  1  OPERATOR_TABLE(ADDR_EXPR_OPTBL,
;  991        P 1121  1          OPERATOR_ENTRY('.', INDIRECT,   PREFIX, 200,  40),
;  992        P 1122  1          OPERATOR_ENTRY('@', INDIRECT,   PREFIX, 200,  40),
;  993        P 1123  1          OPERATOR_ENTRY('+', ADD,        INFIX,   10,  10),
;  994        P 1124  1          OPERATOR_ENTRY('-', SUBTRACT,   INFIX,   10,  10),
;  995        P 1125  1          OPERATOR_ENTRY('+', UNARY_PLUS, PREFIX, 200,  20),
;  996        P 1126  1          OPERATOR_ENTRY('-', UNARY_MINUS,PREFIX, 200,  20),
;  997        P 1127  1          OPERATOR_ENTRY('*', MULTIPLY,   INFIX,   30,  30),
;  998        P 1128  1          OPERATOR_ENTRY('/', DIVIDE,     INFIX,   30,  30),
;  999        P 1129  1          OPERATOR_ENTRY('<', BITSELECT,  POSTFIX, 50, 200, LEXICAL),
; 1000        P 1130  1          OPERATOR_ENTRY('(', OPENPAREN,  PREFIX, 200,   5, LEXICAL),
; 1001          1131  1          OPERATOR_ENTRY(')', CLOSEPAREN, POSTFIX,  6, 200, LEXICAL));
; 1002          1132  1
; 1003          1133  1
; 1004          1134  1  ! Generate the Terminator Lexical Token Entry tables used for the EXAMINE
; 1005          1135  1  ! and other commands (EXAM A,B,C or EXAM A:B,C:D), the DEPOSIT command
; 1006          1136  1  ! (DEP X = Y), the SET BREAK command (SET BREAK X DO(...)), the IF command
; 1007          1137  1  ! (IF X THEN ...), and other situations (no terminator except end of command).
; 1008          1138  1  !
; 1009          1139  1  TERMINATOR_TABLE(EMPTY_TERM_TBL);
; 1010          1140  1
; 1011        P 1141  1  TERMINATOR_TABLE(COMMA_TERM_TBL,
```

```
;  1012          1142  1          TERMINATOR_ENTRY(',', TERM_COMMA));
;  1013          1143  1
;  1014        P 1144  1  TERMINATOR_TABLE(EQUAL_TERM_TBL,
;  1015          1145  1          TERMINATOR_ENTRY('=', TERM_EQUAL, BALANCED_PARENS));
;  1016          1146  1
;  1017        P 1147  1  TERMINATOR_TABLE(DO_TERM_TBL,
;  1018          1148  1          TERMINATOR_ENTRY('DO', TERM_DO, BALANCED_PARENS));
;  1019          1149  1
;  1020        P 1150  1  TERMINATOR_TABLE(THEN_TERM_TBL,
;  1021          1151  1          TERMINATOR_ENTRY('THEN', TERM_THEN, BALANCED_PARENS));
;  1022          1152  1
;  1023        P 1153  1  TERMINATOR_TABLE(COMCOL_TERM_TBL,
;  1024        P 1154  1          TERMINATOR_ENTRY(',', TERM_COMMA),
;  1025          1155  1          TERMINATOR_ENTRY(':', TERM_COLON, MUST_BE_SINGLE));
;  1026          1156  1
;  1027        P 1157  1  TERMINATOR_TABLE(CMWHDO_TERM_TBL,
;  1028        P 1158  1          TERMINATOR_ENTRY(',', TERM_COMMA),
;  1029        P 1159  1          TERMINATOR_ENTRY('WHEN', TERM_WHEN, BALANCED_PARENS),
;  1030          1160  1          TERMINATOR_ENTRY('DO', TERM_DO, BALANCED_PARENS));
;  1031          1161  1
;  1032        P 1162  1  TERMINATOR_TABLE(OPEN_TERM_TBL,
;  1033          1163  1          TERMINATOR_ENTRY('(', TERM_OPEN));
;  1034          1164  1
;  1035        P 1165  1  TERMINATOR_TABLE(COMPAREN_TERM_TBL,
;  1036        P 1166  1          TERMINATOR_ENTRY(',', TERM_COMMA),
;  1037          1167  1          TERMINATOR_ENTRY(')', TERM_CLOSE, BALANCED_PARENS));
;  1038          1168  1
;  1039        P 1169  1  TERMINATOR_TABLE(TO_TERM_TBL,
;  1040          1170  1          TERMINATOR_ENTRY('TO', TERM_TO, BALANCED_PARENS));
;  1041          1171  1
;  1042        P 1172  1  TERMINATOR_TABLE(BY_TERM_TBL,
;  1043        P 1173  1          TERMINATOR_ENTRY('BY', TERM_BY, BALANCED_PARENS),
;  1044          1174  1          TERMINATOR_ENTRY('DO', TERM_DO, BALANCED_PARENS));
;  1045          1175  1
;  1046        P 1176  1  TERMINATOR_TABLE(BIT_SELECT_TERM_TBL,
;  1047        P 1177  1          TERMINATOR_ENTRY(',', TERM_COMMA),
;  1048          1178  1          TERMINATOR_ENTRY('>', TERM_GTRTHAN));
;  1049          1179  1
;  1050        P 1180  1  TERMINATOR_TABLE(SET_CONSTANT_TERM_TBL,
;  1051        P 1181  1          TERMINATOR_ENTRY(',', TERM_COMMA),
;  1052        P 1182  1          TERMINATOR_ENTRY(']', TERM_CLOSE, BALANCED_PARENS),
;  1053          1183  1          TERMINATOR_ENTRY('..', TERM_DOT));
;  1054          1184  1
;  1055          1185  1
;  1056          1186  1  ! Generate a table indexed by Terminator Code which has pointers to the above
;  1057          1187  1  ! Terminator Lexical Token Entry tables.  This table is used in DBG$EXP_IN
;  1058          1188  1  ! and DBG$ADDR_EXP_INT to look up the terminator table to be used for the
;  1059          1189  1  ! current expression of address expression.
;  1060          1190  1  !
;  1061          1191  1  OWN
;  1062          1192  1      TERM_POINTER_TBL: VECTOR[TOKEN$K_MAX_TERMINATOR + 1]
;  1063          1193  1                                  PSECT(DBG$PLIT) PRESET(
;  1064          1194  1          [TOKEN$K_TERM_NONE]    = EMPTY_TERM_TBL          - TABLEBASE,
;  1065          1195  1          [TOKEN$K_TERM_COMMA]   = COMMA_TERM_TBL         - TABLEBASE,
;  1066          1196  1          [TOKEN$K_TERM_EQUAL]   = EQUAL_TERM_TBL         - TABLEBASE,
;  1067          1197  1          [TOKEN$K_TERM_DO]      = DO_TERM_TBL            - TABLEBASE,
;  1068          1198  1          [TOKEN$K_TERM_THEN]    = THEN_TERM_TBL          - TABLEBASE,
```

```
;  1069          1199  1          [TOKEN$K_TERM_COMCOL]  = COMCOL_TERM_TBL          - TABLEBASE,
;  1070          1200  1          [TOKEN$K_TERM_CMWHDO]  = CMWHDO_TERM_TBL          - TABLEBASE,
;  1071          1201  1          [TOKEN$K_TERM_OPEN]    = OPEN_TERM_TBL            - TABLEBASE,
;  1072          1202  1          [TOKEN$K_TERM_COMPAREN]= COMPAREN_TERM_TBL        - TABLEBASE,
;  1073          1203  1          [TOKEN$K_TERM_TO]      = TO_TERM_TBL              - TABLEBASE,
;  1074          1204  1          [TOKEN$K_TERM_BY]      = BY_TERM_TBL              - TABLEBASE);
;  1075          1205  1
;  1076          1206  1
;  1077          1207  1  ! Generate the base Character Table.  The Character Table for each supported
;  1078          1208  1  ! language is generated by using this Character Table as a base and then
;  1079          1209  1  ! specifying a list of exceptions for specific characters.  The result is
;  1080          1210  1  ! a language-specific Character Table, generated in the CHARTBL vector.
;  1081          1211  1  !
;  1082        P 1212  1  CHAR_TABLE(BASE_CHARACTER_TABLE,
;  1083        P 1213  1          CHAR_ENT('A', HEXDIGIT, ALPHABETIC, IDENT_ANYWHERE),
;  1084        P 1214  1          CHAR_ENT('B', B,        ALPHABETIC, IDENT_ANYWHERE, TERMINATOR),
;  1085        P 1215  1          CHAR_ENT('C', HEXDIGIT, ALPHABETIC, IDENT_ANYWHERE),
;  1086        P 1216  1          CHAR_ENT('D', D,        ALPHABETIC, IDENT_ANYWHERE, TERMINATOR),
;  1087        P 1217  1          CHAR_ENT('E', E,        ALPHABETIC, IDENT_ANYWHERE),
;  1088        P 1218  1          CHAR_ENT('F', HEXDIGIT, ALPHABETIC, IDENT_ANYWHERE),
;  1089        P 1219  1          CHAR_ENT('G', G,        ALPHABETIC, IDENT_ANYWHERE),
;  1090        P 1220  1          CHAR_ENT('H', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1091        P 1221  1          CHAR_ENT('I', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1092        P 1222  1          CHAR_ENT('J', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1093        P 1223  1          CHAR_ENT('K', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1094        P 1224  1          CHAR_ENT('L', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1095        P 1225  1          CHAR_ENT('M', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1096        P 1226  1          CHAR_ENT('N', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1097        P 1227  1          CHAR_ENT('O', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1098        P 1228  1          CHAR_ENT('P', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1099        P 1229  1          CHAR_ENT('Q', Q,        ALPHABETIC, IDENT_ANYWHERE),
;  1100        P 1230  1          CHAR_ENT('R', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1101        P 1231  1          CHAR_ENT('S', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1102        P 1232  1          CHAR_ENT('T', OTHER,    ALPHABETIC, IDENT_ANYWHERE, TERMINATOR),
;  1103        P 1233  1          CHAR_ENT('U', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1104        P 1234  1          CHAR_ENT('V', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1105        P 1235  1          CHAR_ENT('W', OTHER,    ALPHABETIC, IDENT_ANYWHERE, TERMINATOR),
;  1106        P 1236  1          CHAR_ENT('X', X,        ALPHABETIC, IDENT_ANYWHERE),
;  1107        P 1237  1          CHAR_ENT('Y', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1108        P 1238  1          CHAR_ENT('Z', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1109        P 1239  1
;  1110        P 1240  1          CHAR_ENT('a', HEXDIGIT, ALP ABETIC, IDENT_ANYWHERE),
;  1111        P 1241  1          CHAR_ENT('b', B,        ALPHABETIC, IDENT_ANYWHERE),
;  1112        P 1242  1          CHAR_ENT('c', HEXDIGIT, ALPHABETIC, IDENT_ANYWHERE),
;  1113        P 1243  1          CHAR_ENT('d', D,        ALPHABETIC, IDENT_ANYWHERE, TERMINATOR),
;  1114        P 1244  1          CHAR_ENT('e', E,        ALPHABETIC, IDENT_ANYWHERE),
;  1115        P 1245  1          CHAR_ENT('f', HEXDIGIT, ALPHABETIC, IDENT_ANYWHERE),
;  1116        P 1246  1          CHAR_ENT('g', G,        ALPHABETIC, IDENT_ANYWHERE),
;  1117        P 1247  1          CHAR_ENT('h', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1118        P 1248  1          CHAR_ENT('i', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1119        P 1249  1          CHAR_ENT('j', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1120        P 1250  1          CHAR_ENT('k', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1121        P 1251  1          CHAR_ENT('l', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1122        P 1252  1          CHAR_ENT('m', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1123        P 1253  1          CHAR_ENT('n', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1124        P 1254  1          CHAR_ENT('o', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1125        P 1255  1          CHAR_ENT('p', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
```

```
;  1126        P 1256  1       CHAR_ENT('q', Q,        ALPHABETIC, IDENT_ANYWHERE),
;  1127        P 1257  1       CHAR_ENT('r', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1128        P 1258  1       CHAR_ENT('s', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1129        P 1259  1       CHAR_ENT('t', OTHER,    ALPHABETIC, IDENT_ANYWHERE, TERMINATOR),
;  1130        P 1260  1       CHAR_ENT('u', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1131        P 1261  1       CHAR_ENT('v', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1132        P 1262  1       CHAR_ENT('w', OTHER,    ALPHABETIC, IDENT_ANYWHERE, TERMINATOR),
;  1133        P 1263  1       CHAR_ENT('x', X,        ALPHABETIC, IDENT_ANYWHERE),
;  1134        P 1264  1       CHAR_ENT('y', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1135        P 1265  1       CHAR_ENT('z', OTHER,    ALPHABETIC, IDENT_ANYWHERE),
;  1136        P 1266  1
;  1137        P 1267  1       CHAR_ENT('0', DIGIT,    DIGIT, IDENT_MIDDLE, IDENT_END, NUMBER_START),
;  1138        P 1268  1       CHAR_ENT('1', DIGIT,    DIGIT, IDENT_MIDDLE, IDENT_END, NUMBER_START),
;  1139        P 1269  1       CHAR_ENT('2', DIGIT,    DIGIT, IDENT_MIDDLE, IDENT_END, NUMBER_START),
;  1140        P 1270  1       CHAR_ENT('3', DIGIT,    DIGIT, IDENT_MIDDLE, IDENT_END, NUMBER_START),
;  1141        P 1271  1       CHAR_ENT('4', DIGIT,    DIGIT, IDENT_MIDDLE, IDENT_END, NUMBER_START),
;  1142        P 1272  1       CHAR_ENT('5', DIGIT,    DIGIT, IDENT_MIDDLE, IDENT_END, NUMBER_START),
;  1143        P 1273  1       CHAR_ENT('6', DIGIT,    DIGIT, IDENT_MIDDLE, IDENT_END, NUMBER_START),
;  1144        P 1274  1       CHAR_ENT('7', DIGIT,    DIGIT, IDENT_MIDDLE, IDENT_END, NUMBER_START),
;  1145        P 1275  1       CHAR_ENT('8', DIGIT,    DIGIT, IDENT_MIDDLE, IDENT_END, NUMBER_START),
;  1146        P 1276  1       CHAR_ENT('9', DIGIT,    DIGIT, IDENT_MIDDLE, IDENT_END, NUMBER_START),
;  1147        P 1277  1
;  1148        P 1278  1       CHAR_ENT(' ', OTHER,    SPACE),
;  1149        P 1279  1       CHAR_ENT(  9, OTHER,    SPACE),         ! Tab character
;  1150        P 1280  1       CHAR_ENT('!', OTHER,    NOTHING),
;  1151        P 1281  1       CHAR_ENT('"', OTHER,    STRING_QUOTE),
;  1152        P 1282  1       CHAR_ENT('#', OTHER,    NOTHING),
;  1153        P 1283  1       CHAR_ENT('$', OTHER,    IDENT_MIDDLE, IDENT_END),
;  1154        P 1284  1       CHAR_ENT('%', OTHER,    NOTHING),
;  1155        P 1285  1       CHAR_ENT('&', OTHER,    NOTHING),
;  1156        P 1286  1       CHAR_ENT('''',OTHER,    STRING_QUOTE),
;  1157        P 1287  1       CHAR_ENT('(', OTHER,    OPCHAR, ADDRESS_OP, TERMINATOR),
;  1158        P 1288  1       CHAR_ENT(')', OTHER,    OPCHAR, ADDRESS_OP, TERMINATOR),
;  1159        P 1289  1       CHAR_ENT('*', OTHER,    OPCHAR, OPCHAR_INFIX, ADDRESS_OP),
;  1160        P 1290  1       CHAR_ENT('+', PLUS,     OPCHAR, ADDRESS_OP),
;  1161        P 1291  1       CHAR_ENT(',', OTHER,    TERMINATOR),
;  1162        P 1292  1       CHAR_ENT('-', MINUS,    OPCHAR, ADDRESS_OP),
;  1163        P 1293  1       CHAR_ENT('.', DOT,      NUMBER_START, OPCHAR, ADDRESS_OP, SPECIAL_SYMBOL),
;  1164        P 1294  1       CHAR_ENT('/', OTHER,    OPCHAR, ADDRESS_OP),
;  1165        P 1295  1       CHAR_ENT(':', OTHER,    TERMINATOR),
;  1166        P 1296  1       CHAR_ENT(';', OTHER,    NOTHING),
;  1167        P 1297  1       CHAR_ENT('<', OTHER,    ADDRESS_OP),
;  1168        P 1298  1       CHAR_ENT('=', OTHER,    TERMINATOR),
;  1169        P 1299  1       CHAR_ENT('>', OTHER,    TERMINATOR),
;  1170        P 1300  1       CHAR_ENT('?', OTHER,    NOTHING),
;  1171        P 1301  1       CHAR_ENT('@', OTHER,    OPCHAR, ADDRESS_OP),
;  1172        P 1302  1       CHAR_ENT('[', OTHER,    NOTHING),
;  1173        P 1303  1       CHAR_ENT('\', OTHER,    OPCHAR, SPECIAL_SYMBOL),
;  1174        P 1304  1       CHAR_ENT(']', OTHER,    NOTHING),
;  1175        P 1305  1       CHAR_ENT('^', OTHER,    SPECIAL_SYMBOL),
;  1176        P 1306  1       CHAR_ENT('_', OTHER,    IDENT_MIDDLE, IDENT_END),
;  1177        P 1307  1       CHAR_ENT('`', OTHER,    NOTHING),
;  1178        P 1308  1       CHAR_ENT('{', OTHER,    NOTHING),
;  1179        P 1309  1       CHAR_ENT('|', OTHER,    NOTHING),
;  1180        P 1310  1       CHAR_ENT('}', OTHER,    NOTHING),
;  1181          1311  1       CHAR_ENT('~', OTHER,    NOTHING));
```

```
;  1183          1312  1 !                L A N G U A G E   U N K N O W N   P A R S E   T A B L E S
;  1184          1313  1 !
;  1185          1314  1 !
;  1186          1315  1 !       This section contains all Lexical Scanner and Parser tables for
;  1187          1316  1 !       language UNKNOWN.  Language "UNKNOWN" constitutes the language
;  1188          1317  1 !       facilities made available by DEBUG when the actual source language
;  1189          1318  1 !       is not known to DEBUG.  The main use for language UNKNOWN is to
;  1190          1319  1 !       provide a basic level of DEBUG support for new languages which are
;  1191          1320  1 !       not yet explicitly supported by DEBUG with its own syntax and
;  1192          1321  1 !       semantics.
;  1193          1322  1 !
;  1194          1323  1 !       (Language UNKNOWN comes first because some of the other languages
;  1195          1324  1 !       make references to the UNKNOWN number table.  The rest of the languages
;  1196          1325  1 !       are in alphabetical order.)
;  1197          1326  1 !
;  1198          1327  1 !
;  1199          1328  1 !
;  1200          1329  1 ! Define the language UNKNOWN Character Table.
;  1201          1330  1 !
;  1202        P 1331  1 CHAR_EXCEPTION_TABLE(UNKNOWN_CHARTBL,
;  1203        P 1332  1         CHAR_ENTRY('&', OTHER,  OPCHAR),
;  1204        P 1333  1         CHAR_ENTRY('/', OTHER,  OPCHAR, OPCHAR_INFIX, ADDRESS_OP),
;  1205        P 1334  1         CHAR_ENTRY('<', OTHER,  OPCHAR, OPCHAR_INFIX, ADDRESS_OP),
;  1206        P 1335  1         CHAR_ENTRY('=', OTHER,  OPCHAR, OPCHAR_INFIX, TERMINATOR),
;  1207        P 1336  1         CHAR_ENTRY('>', OTHER,  OPCHAR, OPCHAR_INFIX, TERMINATOR),
;  1208        P 1337  1         CHAR_ENTRY('^', OTHER,  OPCHAR, SPECIAL_SYMBOL),
;  1209        P 1338  1         CHAR_ENTRY('[', OTHER,  OPCHAR),
;  1210          1339  1         CHAR_ENTRY(']', OTHER,  TERMINATOR));
;  1211          1340  1
;  1212          1341  1
;  1213          1342  1 ! Define the language UNKNOWN Operator Table for operators whose names are
;  1214          1343  1 ! identifiers.
;  1215          1344  1 !
;  1216        P 1345  1 OPERATOR_TABLE(UNKNOWN_IDENT_OPTBL,
;  1217        P 1346  1         OPERATOR_ENTRY('EQL',   EQUAL,          INFIX,   50, 50),
;  1218        P 1347  1         OPERATOR_ENTRY('NEQ',   NOT_EQUAL,      INFIX,   50, 50),
;  1219        P 1348  1         OPERATOR_ENTRY('GTR',   GTR_THAN,       INFIX,   50, 50),
;  1220        P 1349  1         OPERATOR_ENTRY('GEQ',   GTR_EQUAL,      INFIX,   50, 50),
;  1221        P 1350  1         OPERATOR_ENTRY('LSS',   LSS_THAN,       INFIX,   50, 50),
;  1222        P 1351  1         OPERATOR_ENTRY('LEQ',   LSS_EQUAL,      INFIX,   50, 50),
;  1223        P 1352  1         OPERATOR_ENTRY('NOT',   NOT,            PREFIX, 200, 40),
;  1224        P 1353  1         OPERATOR_ENTRY('AND',   AND,            INFIX,   30, 30),
;  1225        P 1354  1         OPERATOR_ENTRY('OR',    OR,             INFIX,   20, 20),
;  1226        P 1355  1         OPERATOR_ENTRY('XOR',   XOR,            INFIX,   10, 10),
;  1227          1356  1         OPERATOR_ENTRY('EQV',   EQV,            INFIX,   10, 10));
;  1228          1357  1
;  1229          1358  1
;  1230          1359  1 ! Define the language UNKNOWN Operator Table for operators whose names are
;  1231          1360  1 ! composed of operator characters such as "+", " " or "*".  This table in-
;  1232          1361  1 ! cludes operators which are part of Primary Symbols (such as "\" and ".").
;  1233          1362  1 !
;  1234        P 1363  1 OPERATOR_TABLE(UNKNOWN_OPCHAR_OPTBL,
;  1235        P 1364  1         OPERATOR_ENTRY('\',     GLOBAL_SLASH,   PREFIX, 0, 0, PRIMARY),
;  1236        P 1365  1         OPERATOR_ENTRY('\',     BACKSLASH,      INFIX,  0, 0, PRIMARY),
;  1237        P 1366  1         OPERATOR_ENTRY('.',     DOT,            INFIX,  0, 0, PRIMARY),
;  1238        P 1367  1         OPERATOR_ENTRY('(',     SUBSCRIPT,      POSTFIX,0, 0, PRIMARY),
;  1239        P 1368  1         OPERATOR_ENTRY('[',     SUBSCRIPT,      POSTFIX,0, 0, PRIMARY),
```

```
;  1240        P 1369  1          OPERATOR_ENTRY('^',     PASCAL_DEREF,   POSTFIX, 0, 0, PRIMARY),
;  1241        P 1370  1
;  1242        P 1371  1          OPERATOR_ENTRY('//',    CONCATENATE,    INFIX,   60,  60),
;  1243        P 1372  1          OPERATOR_ENTRY('&',     CONCATENATE,    INFIX,   60,  60),
;  1244        P 1373  1          OPERATOR_ENTRY('+',     ADD,            INFIX,   60,  60),
;  1245        P 1374  1          OPERATOR_ENTRY('-',     SUBTRACT,       INFIX,   60,  60),
;  1246        P 1375  1          OPERATOR_ENTRY('+',     UNARY_PLUS,     PREFIX, 200,  70),
;  1247        P 1376  1          OPERATOR_ENTRY('-',     UNARY_MINUS,    PREFIX, 200,  70),
;  1248        P 1377  1          OPERATOR_ENTRY('*',     MULTIPLY,       INFIX,   80,  80),
;  1249        P 1378  1          OPERATOR_ENTRY('/',     DIVIDE,         INFIX,   80,  80),
;  1250        P 1379  1          OPERATOR_ENTRY('**',    POWER_OF,       INFIX,   92,  90),
;  1251        P 1380  1          OPERATOR_ENTRY('=',     EQUAL,          INFIX,   50,  50),
;  1252        P 1381  1          OPERATOR_ENTRY('<>',    NOT_EQUAL,      INFIX,   50,  50),
;  1253        P 1382  1          OPERATOR_ENTRY('/=',    NOT_EQUAL,      INFIX,   50,  50),
;  1254        P 1383  1          OPERATOR_ENTRY('>',     GTR_THAN,       INFIX,   50,  50),
;  1255        P 1384  1          OPERATOR_ENTRY('>=',    GTR_EQUAL,      INFIX,   50,  50),
;  1256        P 1385  1          OPERATOR_ENTRY('<',     LSS_THAN,       INFIX,   50,  50),
;  1257        P 1386  1          OPERATOR_ENTRY('<=',    LSS_EQUAL,      INFIX,   50,  50),
;  1258        P 138?  1          OPERATOR_ENTRY('(',     OPENPAREN,      PREFIX, 200,   5, LEXICAL),
;  1259          1388  1          OPERATOR_ENTRY(')',     CLOSEPAREN,     POSTFIX,  6, 200, LEXICAL));
;  1260          1389  1
;  1261          1390  1
;  1262          1391  1  ! Define the UNKNOWN Terminator Lexical Token Table for subscript expressions.
;  1263          1392  1  ! Here we allow subscript expressions to be terminated by ")" (end of sub-
;  1264          1393  1  ! scripts) and by "," (more subscripts to follow).
;  1265          1394  1  !
;  1266        P 1395  1  TERMINATOR_TABLE(UNKNOWN_SUBSCR_TERM_TBL,
;  1267        P 1396  1          TERMINATOR_ENTRY(']', TERM_CLOSE, BALANCED_PARENS),
;  1268        P 1397  1          TERMINATOR_ENTRY(')', TERM_CLOSE, BALANCED_PARENS),
;  1269        P 1398  1          TERMINATOR_ENTRY(':', TERM_COLON, MUST_BE_SINGLE),
;  1270          1399  1          TERMINATOR_ENTRY(',', TERM_COMMA));
;  1271          1400  1
;  1272          1401  1
;  1273          1402  1  ! Define the language UNKNOWN Number Scanner State Table. This is a finite-state
;  1274          1403  1  ! machine in which each transition is of the form:
;  1275          1404  1  !
;  1276          1405  1  !       NUMBER_TRANSITION(character-class, action-index, next-state)
;  1277          1406  1  !
;  1278          1407  1  ! where the character-class and action-index names are automatically prefixed
;  1279          1408  1  ! by 'NUMST$K_CLASS_' or 'NUMST$K_ACT_' by the NUMBER_TRANSITION macro.
;  1280          1409  1  !
;  1281        P 1410  1  NUMBER_STATE_TABLE(UNKNOWN_NUMBER_TABLE,
;  1282        P 1411  1
;  1283        P 1412  1      NUMBER_STATE(START_STATE,
;  1284        P 1413  1          NUMBER_TRANSITION(DIGIT, GO_PAST_DIGIT, ACCUM_INT),
;  1285        P 1414  1          NUMBER_TRANSITION(DOT, MARK_DEC_PT, LEADING_DOT),
;  1286        P 1415  1          NUMBER_TRANSITION(OTHER, NOT_NUMBER, END_STATE)),
;  1287        P 1416  1
;  1288        P 1417  1      NUMBER_STATE(LEADING_DOT,
;  1289        P 1418  1          NUMBER_TRANSITION(DIGIT, GO_PAST_FRAC, ACCUM_FRAC),
;  1290        P 1419  1          NUMBER_TRANSITION(OTHER, NOT_NUMBER, END_STATE)),
;  1291        P 1420  1
;  1292        P 1421  1      NUMBER_STATE(ACCUM_INT,
;  1293        P 1422  1          NUMBER_TRANSITION(DIGIT, GO_PAST_DIGIT, ACCUM_INT),
;  1294        P 1423  1          NUMBER_TRANSITION(DOT, MARK_DEC_PT, ACCUM_FRAC),
;  1295        P 1424  1          NUMBER_TRANSITION(HEXDIGIT, DO_NOTHING, ACCUM_HEX),
;  1296        P 1425  1          NUMBER_TRANSITION(B, DO_NOTHING, ACCUM_HEX),
```

```
;  1297        P 1426  1          NUMBER_TRANSITION(D, DO_NOTHING, ACCUM_HEX),
;  1298        P 1427  1          NUMBER_TRANSITION(E, DO_NOTHING, ACCUM_HEX),
;  1299        P 1428  1          NUMBER_TRANSITION(OTHER, GOT_NUMBER, END_STATE)),
;  1300        P 1429  1
;  1301        P 1430  1      NUMBER_STATE(ACCUM_HEX,
;  1302        P 1431  1          NUMBER_TRANSITION(DIGIT, DO_NOTHING, ACCUM_HEX),
;  1303        P 1432  1          NUMBER_TRANSITION(HEXDIGIT, DO_NOTHING, ACCUM_HEX),
;  1304        P 1433  1          NUMBER_TRANSITION(B, DO_NOTHING, ACCUM_HEX),
;  1305        P 1434  1          NUMBER_TRANSITION(D, DO_NOTHING, ACCUM_HEX),
;  1306        P 1435  1          NUMBER_TRANSITION(E, DO_NOTHING, ACCUM_HEX),
;  1307        P 1436  1          NUMBER_TRANSITION(OTHER, GOT_NUMBER, END_STATE)),
;  1308        P 1437  1
;  1309        P 1438  1      NUMBER_STATE(ACCUM_FRAC,
;  1310        P 1439  1          NUMBER_TRANSITION(DIGIT, GO_PAST_FRAC, ACCUM_FRAC),
;  1311        P 1440  1          NUMBER_TRANSITION(DOT, BACKUP_PTRS, END_STATE),
;  1312        P 1441  1          NUMBER_TRANSITION(E, MARK_E_EXP, GET_EXPONENT),
;  1313        P 1442  1          NUMBER_TRANSITION(D, MARK_D_EXP, GET_EXPONENT),
;  1314        P 1443  1          NUMBER_TRANSITION(G, MARK_G_EXP, GET_EXPONENT),
;  1315        P 1444  1          NUMBER_TRANSITION(Q, MARK_Q_EXP, GET_EXPONENT),
;  1316        P 1445  1          NUMBER_TRANSITION(OTHER, GOT_NUMBER, END_STATE)),
;  1317        P 1446  1
;  1318        P 1447  1      NUMBER_STATE(GET_EXPONENT,
;  1319        P 1448  1          NUMBER_TRANSITION(DIGIT, DO_NOTHING, ACCUM_EXP),
;  1320        P 1449  1          NUMBER_TRANSITION(PLUS, DO_NOTHING, GET_EXP_SIGN),
;  1321        P 1450  1          NUMBER_TRANSITION(MINUS, DO_NOTHING, GET_EXP_SIGN),
;  1322        P 1451  1          NUMBER_TRANSITION(OTHER, BACKUP_PTRS, END_STATE)),
;  1323        P 1452  1
;  1324        P 1453  1      NUMBER_STATE(GET_EXP_SIGN,
;  1325        P 1454  1          NUMBER_TRANSITION(DIGIT, DO_NOTHING, ACCUM_EXP),
;  1326        P 1455  1          NUMBER_TRANSITION(OTHER, GOT_NUMBER, END_STATE)),
;  1327        P 1456  1
;  1328        P 1457  1      NUMBER_STATE(ACCUM_EXP,
;  1329        P 1458  1          NUMBER_TRANSITION(DIGIT, DO_NOTHING, ACCUM_EXP),
;  1330        P 1459  1          NUMBER_TRANSITION(OTHER, GOT_NUMBER, END_STATE)),
;  1331        P 1460  1
;  1332        P 1461  1      NUMBER_STATE(END_STATE,
;  1333          1462  1          NUMBER_TRANSITION(OTHER, GIVE_ERROR, END_STATE)));
;  1334          1463  1
;  1335          1464  1
;  1336          1465  1  ! Define the language UNKNOWN Predefined Identifier Table.
;  1337          1466  1  !
;  1338          1467  1  PRID_TABLE(UNKNOWN_PRID_TABLE);
;  1339          1468  1
;  1340          1469  1
;  1341          1470  1  ! Define the language UNKNOWN Built-in Function Table.
;  1342          1471  1  !
;  1343          1472  1  BUILT_IN_FUNCTION_TABLE(UNKNOWN_FUNCTION_TABLE);
;  1344          1473  1
;  1345          1474  1
;  1346          1475  1  ! Define the Primary Parser State Table for language UNKNOWN.
;  1347          1476  1  !
;  1348        P 1477  1  PRIMARY_STATE_TABLE(UNKNOWN_PRIMARY_TABLE,
;  1349        P 1478  1
;  1350        P 1479  1      PRIMARY_STATE(START_STATE,
;  1351        P 1480  1          PRIMARY_TRANSITION(GLOBAL_SLASH, START_GBL, GET_GLOBAL),
;  1352        P 1481  1          PRIMARY_TRANSITION(BACKSLASH, START_SLASH, GOT_BACKSLASH),
;  1353        P 1482  1          PRIMARY_TRANSITION(INVOCNUM, SLASH_INVOCNUM, GOT_BACKSLASH),
```

```
;  1354        P 1483  1          PRIMARY_TRANSITION(DOT, START_DOT, GOT_DOT),
;  1355        P 1484  1          PRIMARY_TRANSITION(SUBSCRIPT, START_SUBSCR, GOT_SUBSCRIPT),
;  1356        P 1485  1          PRIMARY_TRANSITION(PASCAL_DEREF, START_DEREF, GOT_DEREF),
;  1357        P 1486  1          PRIMARY_TRANSITION(PRIMARY_TERM, START_TERM, END_STATE)),
;  1358        P 1487  1
;  1359        P 1488  1      PRIMARY_STATE(GET_GLOBAL,
;  1360        P 1489  1          PRIMARY_TRANSITION(PRIMARY_TERM, GBL_TERM, END_STATE)),
;  1361        P 1490  1
;  1362        P 1491  1      PRIMARY_STATE(GOT_BACKSLASH,
;  1363        P 1492  1          PRIMARY_TRANSITION(BACKSLASH, SLASH_SLASH, GOT_BACKSLASH),
;  1364        P 1493  1          PRIMARY_TRANSITION(INVOCNUM, SLASH_INVOCNUM, GOT_BACKSLASH),
;  1365        P 1494  1          PRIMARY_TRANSITION(DOT, SLASH_DOT, GOT_DOT),
;  1366        P 1495  1          PRIMARY_TRANSITION(SUBSCRIPT, SLASH_SUBSCR, GOT_SUBSCRIPT),
;  1367        P 1496  1          PRIMARY_TRANSITION(PASCAL_DEREF, SLASH_DEREF, GOT_DEREF),
;  1368        P 1497  1          PRIMARY_TRANSITION(PRIMARY_TERM, SLASH_TERM, END_STATE)),
;  1369        P 1498  1
;  1370        P 1499  1      PRIMARY_STATE(GOT_DOT,
;  1371        P 1500  1          PRIMARY_TRANSITION(DOT, DOT_DOT, GOT_DOT),
;  1372        P 1501  1          PRIMARY_TRANSITION(SUBSCRIPT, DOT_SUBSCR, GOT_SUBSCRIPT),
;  1373        P 1502  1          PRIMARY_TRANSITION(PASCAL_DEREF, DOT_DEREF, GOT_DEREF),
;  1374        P 1503  1          PRIMARY_TRANSITION(PRIMARY_TERM, DOT_TERM, END_STATE)),
;  1375        P 1504  1
;  1376        P 1505  1      PRIMARY_STATE(GOT_SUBSCRIPT,
;  1377        P 1506  1          PRIMARY_TRANSITION(DOT, SUBSCR_DOT, GOT_DOT),
;  1378        P 1507  1          PRIMARY_TRANSITION(SUBSCRIPT, SUBSCR_SUBSCR, GOT_SUBSCRIPT),
;  1379        P 1508  1          PRIMARY_TRANSITION(PASCAL_DEREF, SUBSCR_DEREF, GOT_DEREF),
;  1380        P 1509  1          PRIMARY_TRANSITION(PRIMARY_TERM, SUBSCR_TERM, END_STATE)),
;  1381        P 1510  1
;  1382        P 1511  1      PRIMARY_STATE(GOT_DEREF,
;  1383        P 1512  1          PRIMARY_TRANSITION(DOT, DEREF_DOT, GOT_DOT),
;  1384        P 1513  1          PRIMARY_TRANSITION(SUBSCRIPT, DEREF_SUBSCR, GOT_SUBSCRIPT),
;  1385        P 1514  1          PRIMARY_TRANSITION(PASCAL_DEREF, DEREF_DEREF, GOT_DEREF),
;  1386        P 1515  1          PRIMARY_TRANSITION(PRIMARY_TERM, DEREF_TERM, END_STATE)),
;  1387        P 1516  1
;  1388          1517  1      PRIMARY_STATE(END_STATE));
;  1389          1518  1
;  1390          1519  1
;  1391          1520  1  ! Finally define the table of pointers to the parse tables for language UNKNOWN.
;  1392          1521  1  !
;  1393        P 1522  1  LANGUAGE_TABLES(LANGUAGE = UNKNOWN,
;  1394        P 1523  1                  CHARTBL = UNKNOWN_CHARTBL,
;  1395        P 1524  1                  IDENT_OPTBL = UNKNOWN_IDENT_OPTBL,
;  1396        P 1525  1                  OPCHAR_OPTBL = UNKNOWN_OPCHAR_OPTBL,
;  1397        P 1526  1                  NUMBER_TABLE = UNKNOWN_NUMBER_TABLE,
;  1398        P 1527  1                  PRIMARY_TABLE = UNKNOWN_PRIMARY_TABLE,
;  1399        P 1528  1                  SUBSCR_TERMS = UNKNOWN_SUBSCR_TERM_TBL,
;  1400        P 1529  1                  PRIDTBL = UNKNOWN_PRID_TABLE,
;  1401          1530  1                  BIF_TABLE = UNKNOWN_FUNCTION_TABLE);
```

```
;  1403          1531  1 !                          A D A   P A R S E   T A B L E S
;  1404          1532  1 !
;  1405          1533  1 !
;  1406          1534  1 !
;  1407          1535  1 !       This section includes all the Lexical Scanner and Parser tables needed
;  1408          1536  1 !       to scan and parse the ADA language.
;  1409          1537  1 !
;  1410          1538  1 !
;  1411          1539  1 !
;  1412          1540  1 ! Define the ADA Character Table.  What is listed here is actually a list of
;  1413          1541  1 ! exceptions to the Character Table for Language UNKNOWN.
;  1414          1542  1 !
;  1415        P 1543  1 CHAR_EXCEPTION_TABLE(ADA_CHARTBL,
;  1416        P 1544  1         CHAR_ENTRY('''', OTHER,      NOTHING),
;  1417        P 1545  1         CHAR_ENTRY('_', UNDERSCORE, IDENT_MIDDLE),
;  1418        P 1546  1         CHAR_ENTRY('#', POUND,      NOTHING),
;  1419        P 1547  1         CHAR_ENTRY('&', OTHER,      OPCHAR),
;  1420        P 1548  1         CHAR_ENTRY('/', OTHER,      OPCHAR, OPCHAR_INFIX, ADDRESS_OP),
;  1421        P 1549  1         CHAR_ENTRY('<', OTHER,      OPCHAR, OPCHAR_INFIX, ADDRESS_OP),
;  1422        P 1550  1         CHAR_ENTRY('=', OTHER,      OPCHAR, OPCHAR_INFIX, TERMINATOR),
;  1423        P 1551  1         CHAR_ENTRY('>', OTHER,      OPCHAR, OPCHAR_INFIX, TERMINATOR),
;  1424          1552  1         CHAR_ENTRY('.', DOT,        OPCHAR, ADDRESS_OP, SPECIAL_SYMBOL, TERMINATOR));
;  1425          1553  1
;  1426          1554  1
;  1427          1555  1 ! Define the ADA Operator Table for operators whose names are identifiers.
;  1428          1556  1 !
;  1429        P 1557  1 OPERATOR_TABLE(ADA_IDENT_OPTBL,
;  1430        P 1558  1         OPERATOR_ENTRY('NOT',    NOT,          PREFIX, 200, 60),
;  1431        P 1559  1         OPERATOR_ENTRY('ABS',    ABSOLUTE,     PREFIX, 200, 60),
;  1432        P 1560  1         OPERATOR_ENTRY('MOD',    MODULUS,      INFIX,   50, 50),
;  1433        P 1561  1         OPERATOR_ENTRY('REM',    REMAINDER,    INFIX,   50, 50),
;  1434        P 1562  1         OPERATOR_ENTRY('AND',    AND,          INFIX,   10, 10),
;  1435        P 1563  1         OPERATOR_ENTRY('OR',     OR,           INFIX,   10, 10),
;  1436          1564  1         OPERATOR_ENTRY('XOR',    XOR,          INFIX,   10, 10));
;  1437          1565  1
;  1438          1566  1
;  1439          1567  1 ! Define the ADA Operator Table for operators whose names are composed of
;  1440          1568  1 ! operator characters such as "+", "-", or "*".  This table includes operators
;  1441          1569  1 ! which are part of DEBUG Primary Symbols (such as "\").
;  1442          1570  1 !
;  1443        P 1571  1 OPERATOR_TABLE(ADA_OPCHAR_OPTBL,
;  1444        P 1572  1         OPERATOR_ENTRY('\',      GLOBAL_SLASH,  PREFIX, 0, 0, PRIMARY),
;  1445        P 1573  1         OPERATOR_ENTRY('\',      BACKSLASH,     INFIX,  0, 0, PRIMARY),
;  1446        P 1574  1         OPERATOR_ENTRY('.',      DOT,           INFIX,  0, 0, PRIMARY),
;  1447        P 1575  1         OPERATOR_ENTRY('(',      SUBSCRIPT,     POSTFIX,0, 0, PRIMARY),
;  1448        P 1576  1
;  1449        P 1577  1         OPERATOR_ENTRY('(',      OPENPAREN,     PREFIX, 200,   5, LEXICAL),
;  1450        P 1578  1         OPERATOR_ENTRY(')',      CLOSEPAREN,    POSTFIX,  6, 200, LEXICAL),
;  1451        P 1579  1         OPERATOR_ENTRY('**',     POWER_OF,      INFIX,   60,  60),
;  1452        P 1580  1         OPERATOR_ENTRY('*',      MULTIPLY,      INFIX,   50,  50),
;  1453        P 1581  1         OPERATOR_ENTRY('/',      DIVIDE,        INFIX,   50,  50),
;  1454        P 1582  1         OPERATOR_ENTRY('+',      UNARY_PLUS,    PREFIX, 200,  40),
;  1455        P 1583  1         OPERATOR_ENTRY('-',      UNARY_MINUS,   PREFIX, 200,  40),
;  1456        P 1584  1         OPERATOR_ENTRY('+',      ADD,           INFIX,   30,  30),
;  1457        P 1585  1         OPERATOR_ENTRY('-',      SUBTRACT,      INFIX,   30,  30),
;  1458        P 1586  1         OPERATOR_ENTRY('&',      CONCATENATE,   INFIX,   30,  30),
;  1459        P 1587  1         OPERATOR_ENTRY('=',      EQUAL,         INFIX,   20,  20),
```

```
;  1460        P 1588  1          OPERATOR_ENTRY('/=',    NOT_EQUAL,      INFIX,   20,  20),
;  1461        P 1589  1          OPERATOR_ENTRY('<',     LSS_THAN,       INFIX,   20,  20),
;  1462        P 1590  1          OPERATOR_ENTRY('<=',    LSS_EQUAL,      INFIX,   20,  20),
;  1463        P 1591  1          OPERATOR_ENTRY('>',     GTR_THAN,       INFIX,   20,  20),
;  1464          1592  1          OPERATOR_ENTRY('>=',    GTR_EQUAL,      INFIX,   20,  20));
;  1465          1593  1
;  1466          1594  1
;  1467          1595  1  ! Define an Operator Lexical Token Entry for the ADA Tick operator "'".
;  1468          1596  1  ! This token is actually used to represent a collection of postfix
;  1469          1597  1  ! tick operators ("'FIRST", "'LAST", ...). The lexical scanner will
;  1470          1598  1  ! fill in the subcode field which identifies which tick operator was
;  1471          1599  1  ! given.
;  1472          1600  1  !
;  1473          1601  1  BIND
;  1474          1602  1      ADA_TICK_TOKEN = OPERATOR_ENTRY('''', ADA_TICK, POSTFIX, 0, 0, PRIMARY);
;  1475          1603  1
;  1476          1604  1
;  1477          1605  1  OWN
;  1478          1606  1      ADA_TICK_TABLE: VECTOR [TOKEN$K_TICK_MAX+1] PSECT(DBG$PLIT) PRESET (
;  1479          1607  1          [TOKEN$K_TICK_CONSTRAINED]      = UPLIT (%ASCIC 'CONSTRAINED')  - TABLEBASE,
;  1480          1608  1          [TOKEN$K_TICK_FIRST]            = UPLIT (%ASCIC 'FIRST')        - TABLEBASE,
;  1481          1609  1          [TOKEN$K_TICK_LAST]             = UPLIT (%ASCIC 'LAST')         - TABLEBASE,
;  1482          1610  1          [TOKEN$K_TICK_LENGTH]           = UPLIT (%ASCIC 'LENGTH')       - TABLEBASE,
;  1483          1611  1          [TOKEN$K_TICK_POS]              = UPLIT (%ASCIC 'POS')          - TABLEBASE,
;  1484          1612  1          [TOKEN$K_TICK_PRED]             = UPLIT (%ASCIC 'PRED')         - TABLEBASE,
;  1485          1613  1          [TOKEN$K_TICK_SIZE]             = UPLIT (%ASCIC 'SIZE')         - TABLEBASE,
;  1486          1614  1          [TOKEN$K_TICK_SUCC]             = UPLIT (%ASCIC 'SUCC')         - TABLEBASE,
;  1487          1615  1          [TOKEN$K_TICK_VAL]              = UPLIT (%ASCIC 'VAL')          - TABLEBASE);
;  1488          1616  1
;  1489          1617  1  ! Define the ADA Terminator Lexical Token Table for subscript expressions.
;  1490          1618  1  !
;  1491        P 1619  1  TERMINATOR_TABLE(ADA_SUBSCR_TERM_TBL,
;  1492        P 1620  1          TERMINATOR_ENTRY(')', TERM_CLOSE),
;  1493        P 1621  1          TERMINATOR_ENTRY(':', TERM_COLON, MUST_BE_SINGLE),
;  1494        P 1622  1          TERMINATOR_ENTRY('..',TERM_COLON),
;  1495          1623  1          TERMINATOR_ENTRY(',', TERM_COMMA));
;  1496          1624  1
;  1497          1625  1
;  1498          1626  1  ! Define the ADA Predefined Identifier Table.
;  1499          1627  1  !
;  1500          1628  1  PRID_TABLE(ADA_PRID_TABLE);
;  1501          1629  1
;  1502          1630  1
;  1503          1631  1  ! Define the ADA Built-in Function Table.
;  1504          1632  1  !
;  1505          1633  1  BUILT_IN_FUNCTION_TABLE(ADA_FUNCTION_TABLE);
;  1506          1634  1
;  1507          1635  1
;  1508          1636  1  ! Define the ADA Number Scanner State Table.  This table defines the states
;  1509          1637  1  ! of a Finite-State Machine which picks up all valid numeric constants in the
;  1510          1638  1  ! language.
;  1511          1639  1  !
;  1512          1640  1  ! Each transition is of the form:
;  1513          1641  1  !       NUMBER_TRANSITION(character-class, action-index, next-state)
;  1514          1642  1  !
;  1515          1643  1  ! The ADA standard defines a number to be of the form:
;  1516          1644  1  !
```

```
;1517          1645  1 !       number_literal ::=  integer [.integer] [exponent]
;1518          1646  1 !       integer ::= digit { [underline] digit }
;1519          1647  1 !       exponent ::= E [+] integer ! E - integer
;1520          1648  1 !
;1521          1649  1 ! Examples:
;1522          1650  1 !
;1523          1651  1 !       12      1E6         123_456         integer literals
;1524          1652  1 !       12.0    3.141_592 1.34E+6           real literals
;1525          1653  1 !
;1526          1654  1 ! The ADA standard also defines a based number to be of the form:
;1527          1655  1 !
;1528          1656  1 !       base # based_integer [.based_integer] # [exponent]
;1529          1657  1 !       base ::= integer
;1530          1658  1 !       based_integer ::= extended_digit { [underline] extended_digit }
;1531          1659  1 !       extended_digit ::= digit ! letter
;1532          1660  1 !
;1533          1661  1 ! Examples:
;1534          1662  1 !
;1535          1663  1 !       2#1111_1111#  16#FF#              integer literals
;1536          1664  1 !       16#FF.FF#E+2                      real literal
;1537          1665  1 !
;1538        P 1666  1 NUMBER_STATE_TABLE(ADA_NUMBER_TABLE,
;1539        P 1667  1
;1540        P 1668  1     NUMBER_STATE(START_STATE,
;1541        P 1669  1         NUMBER_TRANSITION(DIGIT, DO_NOTHING, ACCUM_INT),
;1542        P 1670  1         NUMBER_TRANSITION(OTHER, NOT_NUMBER, END_STATE)),
;1543        P 1671  1
;1544        P 1672  1     NUMBER_STATE(ACCUM_INT,
;1545        P 1673  1         NUMBER_TRANSITION(DIGIT, DO_NOTHING, ACCUM_INT),
;1546        P 1674  1         NUMBER_TRANSITION(UNDERSCORE, DO_NOTHING, T_ACCUM_INT),
;1547        P 1675  1         NUMBER_TRANSITION(POUND, SAVE_BASE, B_START_STATE),
;1548        P 1676  1         NUMBER_TRANSITION(DOT, MARK_DEC_PT, ACCUM_FRAC),
;1549        P 1677  1         NUMBER_TRANSITION(E, DO_NOTHING, GET_EXPONENT),
;1550        P 1678  1         NUMBER_TRANSITION(OTHER, GOT_NUMBER, END_STATE)),
;1551        P 1679  1
;1552        P 1680  1     NUMBER_STATE(T_ACCUM_INT,
;1553        P 1681  1         NUMBER_TRANSITION(DIGIT, DO_NOTHING, ACCUM_INT),
;1554        P 1682  1         NUMBER_TRANSITION(OTHER, NOT_NUMBER, END_STATE)),
;1555        P 1683  1
;1556        P 1684  1     NUMBER_STATE(ACCUM_FRAC,
;1557        P 1685  1         NUMBER_TRANSITION(DIGIT, DO_NOTHING, ACCUM_FRAC),
;1558        P 1686  1         NUMBER_TRANSITION(UNDERSCORE, DO_NOTHING, T_ACCUM_FRAC),
;1559        P 1687  1         NUMBER_TRANSITION(DOT, NOT_NUMBER, END_STATE),
;1560        P 1688  1         NUMBER_TRANSITION(E, MARK_E_EXP, GET_EXPONENT),
;1561        P 1689  1         NUMBER_TRANSITION(D, MARK_D_EXP, GET_EXPONENT),
;1562        P 1690  1         NUMBER_TRANSITION(G, MARK_G_EXP, GET_EXPONENT),
;1563        P 1691  1         NUMBER_TRANSITION(Q, MARK_Q_EXP, GET_EXPONENT),
;1564        P 1692  1         NUMBER_TRANSITION(OTHER, GOT_NUMBER, END_STATE)),
;1565        P 1693  1
;1566        P 1694  1     NUMBER_STATE(T_ACCUM_FRAC,
;1567        P 1695  1         NUMBER_TRANSITION(DIGIT, DO_NOTHING, ACCUM_FRAC),
;1568        P 1696  1         NUMBER_TRANSITION(OTHER, NOT_NUMBER, END_STATE)),
;1569        P 1697  1
;1570        P 1698  1     NUMBER_STATE(GET_EXPONENT,
;1571        P 1699  1         NUMBER_TRANSITION(DIGIT, DO_NOTHING, ACCUM_EXP),
;1572        P 1700  1         NUMBER_TRANSITION(PLUS, DO_NOTHING, GET_EXP_SIGN),
;1573        P 1701  1         NUMBER_TRANSITION(MINUS, DO_NOTHING, GET_EXP_SIGN),
```

```
;  1574        P 1702  1          NUMBER_TRANSITION(OTHER, NOT_NUMBER, END_STATE)),
;  1575        P 1703  1
;  1576        P 1704  1      NUMBER_STATE(GET_EXP_SIGN,
;  1577        P 1705  1          NUMBER_TRANSITION(DIGIT, DO_NOTHING, ACCUM_EXP),
;  1578        P 1706  1          NUMBER_TRANSITION(OTHER, NOT_NUMBER, END_STATE)),
;  1579        P 1707  1
;  1580        P 1708  1      NUMBER_STATE(ACCUM_EXP,
;  1581        P 1709  1          NUMBER_TRANSITION(DIGIT, DO_NOTHING, ACCUM_EXP),
;  1582        P 1710  1          NUMBER_TRANSITION(UNDERSCORE, DO_NOTHING, T_ACCUM_EXP),
;  1583        P 1711  1          NUMBER_TRANSITION(OTHER, GOT_NUMBER, END_STATE)),
;  1584        P 1712  1
;  1585        P 1713  1      NUMBER_STATE(T_ACCUM_EXP,
;  1586        P 1714  1          NUMBER_TRANSITION(DIGIT, DO_NOTHING, ACCUM_EXP),
;  1587        P 1715  1          NUMBER_TRANSITION(OTHER, NOT_NUMBER, END_STATE)),
;  1588        P 1716  1
;  1589        P 1717  1      NUMBER_STATE(B_START_STATE,
;  1590        P 1718  1          NUMBER_TRANSITION(DIGIT, DO_NOTHING, B_ACCUM_INT),
;  1591        P 1719  1          NUMBER_TRANSITION(OTHER, NOT_NUMBER, END_STATE)),
;  1592        P 1720  1
;  1593        P 1721  1      NUMBER_STATE(B_ACCUM_INT,
;  1594        P 1722  1          NUMBER_TRANSITION(DIGIT, DO_NOTHING, B_ACCUM_INT),
;  1595        P 1723  1          NUMBER_TRANSITION(HEXDIGIT, DO_NOTHING, B_ACCUM_INT),
;  1596        P 1724  1          NUMBER_TRANSITION(B, DO_NOTHING, B_ACCUM_INT),
;  1597        P 1725  1          NUMBER_TRANSITION(D, DO_NOTHING, B_ACCUM_INT),
;  1598        P 1726  1          NUMBER_TRANSITION(E, DO_NOTHING, B_ACCUM_INT),
;  1599        P 1727  1          NUMBER_TRANSITION(UNDERSCORE, DO_NOTHING, T_B_ACCUM_INT),
;  1600        P 1728  1          NUMBER_TRANSITION(DOT, MARK_DEC_PT, B_ACCUM_FRAC),
;  1601        P 1729  1          NUMBER_TRANSITION(POUND, DO_NOTHING, GET_EXPONENT),
;  1602        P 1730  1          NUMBER_TRANSITION(OTHER, GOT_NUMBER, END_STATE)),
;  1603        P 1731  1
;  1604        P 1732  1      NUMBER_STATE(T_B_ACCUM_INT,
;  1605        P 1733  1          NUMBER_TRANSITION(DIGIT, DO_NOTHING, B_ACCUM_INT),
;  1606        P 1734  1          NUMBER_TRANSITION(HEXDIGIT, DO_NOTHING, B_ACCUM_INT),
;  1607        P 1735  1          NUMBER_TRANSITION(B, DO_NOTHING, B_ACCUM_INT),
;  1608        P 1736  1          NUMBER_TRANSITION(D, DO_NOTHING, B_ACCUM_INT),
;  1609        P 1737  1          NUMBER_TRANSITION(E, DO_NOTHING, B_ACCUM_INT),
;  1610        P 1738  1          NUMBER_TRANSITION(OTHER, NOT_NUMBER, END_STATE)),
;  1611        P 1739  1
;  1612        P 1740  1      NUMBER_STATE(B_ACCUM_FRAC,
;  1613        P 1741  1          NUMBER_TRANSITION(DIGIT, DO_NOTHING, B_ACCUM_FRAC),
;  1614        P 1742  1          NUMBER_TRANSITION(HEXDIGIT, DO_NOTHING, B_ACCUM_FRAC),
;  1615        P 1743  1          NUMBER_TRANSITION(B, DO_NOTHING, B_ACCUM_FRAC),
;  1616        P 1744  1          NUMBER_TRANSITION(D, DO_NOTHING, B_ACCUM_FRAC),
;  1617        P 1745  1          NUMBER_TRANSITION(E, DO_NOTHING, B_ACCUM_FRAC),
;  1618        P 1746  1          NUMBER_TRANSITION(UNDERSCORE, DO_NOTHING, T_B_ACCUM_FRAC),
;  1619        P 1747  1          NUMBER_TRANSITION(POUND, MARK_E_EXP, GET_EXPONENT),
;  1620        P 1748  1          NUMBER_TRANSITION(OTHER, NOT_NUMBER, END_STATE)),
;  1621        P 1749  1
;  1622        P 1750  1      NUMBER_STATE(T_B_ACCUM_FRAC,
;  1623        P 1751  1          NUMBER_TRANSITION(DIGIT, DO_NOTHING, B_ACCUM_FRAC),
;  1624        P 1752  1          NUMBER_TRANSITION(HEXDIGIT, DO_NOTHING, B_ACCUM_FRAC),
;  1625        P 1753  1          NUMBER_TRANSITION(B, DO_NOTHING, B_ACCUM_FRAC),
;  1626        P 1754  1          NUMBER_TRANSITION(D, DO_NOTHING, B_ACCUM_FRAC),
;  1627        P 1755  1          NUMBER_TRANSITION(E, DO_NOTHING, B_ACCUM_FRAC),
;  1628        P 1756  1          NUMBER_TRANSITION(OTHER, NOT_NUMBER, END_STATE)),
;  1629        P 1757  1
;  1630        P 1758  1      NUMBER_STATE(END_STATE,
```

```
;  1631          1759  1          NUMBER_TRANSITION(OTHER, GIVE_ERROR, END_STATE)));
;  1632          1760  1
;  1633          1761  1
;  1634          1762  1  ! Save away the value of the B_START_STATE state for ADA. This will be
;  1635          1763  1  ! used later in the number scanner.
;  1636          1764  1  !
;  1637          1765  1  COMPILETIME
;  1638          1766  1      REMEMBER_ADA_B_START_STATE = 0;
;  1639          1767  1  %ASSIGN (REMEMBER_ADA_B_START_STATE, NUMST$XX_STATE_B_START_STATE);
;  1640          1768  1
;  1641          1769  1
;  1642          1770  1  ! Define the Primary Parser State Table for language ADA.  Each transition
;  1643          1771  1  ! Entry in the state table has this format:
;  1644          1772  1  !
;  1645          1773  1  !       PRIMARY_TRANSITION(operator-code, action, next-state)
;  1646          1774  1  !
;  1647          1775  1  ! where the first parameter is the operator code which causes the transition
;  1648          1776  1  ! to be taken, the second parameter is the action routine CASE index for the
;  1649          1777  1  ! transition, and the third parameter is the next state in the Finite-State
;  1650          1778  1  ! Machine.
;  1651          1779  1  !
;  1652        P 1780  1  PRIMARY_STATE_TABLE(ADA_PRIMARY_TABLE,
;  1653        P 1781  1
;  1654        P 1782  1      PRIMARY_STATE(START_STATE,
;  1655        P 1783  1          PRIMARY_TRANSITION(GLOBAL_SLASH, START_GBL, GET_GLOBAL),
;  1656        P 1784  1          PRIMARY_TRANSITION(BACKSLASH, START_SLASH, GOT_BACKSLASH),
;  1657        P 1785  1          PRIMARY_TRANSITION(INVOCNUM, SLASH_INVOCNUM, GOT_BACKSLASH),
;  1658        P 1786  1          PRIMARY_TRANSITION(DOT, START_DOT, GOT_DOT),
;  1659        P 1787  1          PRIMARY_TRANSITION(SUBSCRIPT, START_SUBSCR, GOT_SUBSCRIPT),
;  1660        P 1788  1          PRIMARY_TRANSITION(ADA_TICK, START_TICK, END_STATE),
;  1661        P 1789  1          PRIMARY_TRANSITION(PRIMARY_TERM, START_TERM, END_STATE)),
;  1662        P 1790  1
;  1663        P 1791  1      PRIMARY_STATE(GET_GLOBAL,
;  1664        P 1792  1          PRIMARY_TRANSITION(PRIMARY_TERM, GBL_TERM, END_STATE)),
;  1665        P 1793  1
;  1666        P 1794  1      PRIMARY_STATE(GOT_BACKSLASH,
;  1667        P 1795  1          PRIMARY_TRANSITION(BACKSLASH, SLASH_SLASH, GOT_BACKSLASH),
;  1668        P 1796  1          PRIMARY_TRANSITION(INVOCNUM, SLASH_INVOCNUM, GOT_BACKSLASH),
;  1669        P 1797  1          PRIMARY_TRANSITION(DOT, SLASH_DOT, GOT_DOT),
;  1670        P 1798  1          PRIMARY_TRANSITION(SUBSCRIPT, SLASH_SUBSCR, GOT_SUBSCRIPT),
;  1671        P 1799  1          PRIMARY_TRANSITION(ADA_TICK, SLASH_TICK, END_STATE),
;  1672        P 1800  1          PRIMARY_TRANSITION(PRIMARY_TERM, SLASH_TERM, END_STATE)),
;  1673        P 1801  1
;  1674        P 1802  1      PRIMARY_STATE(GOT_DOT,
;  1675        P 1803  1          PRIMARY_TRANSITION(DOT, DOT_DOT, GOT_DOT),
;  1676        P 1804  1          PRIMARY_TRANSITION(SUBSCRIPT, DOT_SUBSCR, GOT_SUBSCRIPT),
;  1677        P 1805  1          PRIMARY_TRANSITION(ADA_TICK, DOT_TICK, END_STATE),
;  1678        P 1806  1          PRIMARY_TRANSITION(PRIMARY_TERM, DOT_TERM, END_STATE)),
;  1679        P 1807  1
;  1680        P 1808  1      PRIMARY_STATE(GOT_SUPSCRIPT,
;  1681        P 1809  1          PRIMARY_TRANSITION(DOT, SUBSCR_DOT, GOT_DOT),
;  1682        P 1810  1          PRIMARY_TRANSITION(SUBSCRIPT, SUBSCR_SUBSCR, GOT_SUBSCRIPT),
;  1683        P 1811  1          PRIMARY_TRANSITION(PRIMARY_TERM, SUBSCR_TERM, END_STATE)),
;  1684        P 1812  1
;  1685          1813  1      PRIMARY_STATE(END_STATE));
;  1686          1814  1
;  1687          1815  1
```

```
;1688          1816  1 ! Define the table of pointers to the parse tables for ADA.
;1689          1817  1 !
;1690        P 1818  1 LANGUAGE_TABLES(LANGUAGE = ADA,
;1691        P 1819  1                 CHARTBL = ADA_CHARTBL,
;1692        P 1820  1                 IDENT_OPTBL = ADA_IDENT_OPTBL,
;1693        P 1821  1                 OPCHAR_OPTBL = ADA_OPCHAR_OPTBL,
;1694        P 1822  1                 NUMBER_TABLE = ADA_NUMBER_TABLE,
;1695        P 1823  1                 PRIMARY_TABLE = ADA_PRIMARY_TABLE,
;1696        P 1824  1                 SUBSCR_TERMS = ADA_SUBSCR_TERM_TBL,
;1697        P 1825  1                 PRIDTBL = ADA_PRID_TABLE,
;1698          1826  1                 BIF_TABLE = ADA_FUNCTION_TABLE);
```

```
;1700          1827  1 !                          B A S I C   P A R S E   T A B L E S
;1701          1828  1 !
;1702          1829  1 !
;1703          1830  1 !
;1704          1831  1 !       This section includes all the Lexical Scanner and Parser tables needed
;1705          1832  1 !       to scan and parse BASIC expressions.
;1706          1833  1 !
;1707          1834  1 !
;1708          1835  1 !
;1709          1836  1 ! Define the BASIC Character Table.  What is listed here is actually a list
;1710          1837  1 ! of exceptions to the Character Table for language UNKNOWN.  (Language UNKNOWN
;1711          1838  1 ! lists the "average" use of each character in the character set.)
;1712          1839  1 !
;1713        P 1840  1 CHAR_EXCEPTION_TABLE(BASIC_CHARTBL,
;1714        P 1841  1         CHAR_ENTRY('.', DOT,    NUMBER_START, IDENT_MIDDLE, IDENT_END, ADDRESS_OP, SPECIAL_SYMBOL),
;1715        P 1842  1         CHAR_ENTRY('%', OTHER,  IDENT_END),
;1716        P 1843  1         CHAR_ENTRY(':', OTHER,  OPCHAR, OPCHAR_INFIX, TERMINATOR),
;1717        P 1844  1         CHAR_ENTRY('^', OTHER,  OPCHAR, SPECIAL_SYMBOL),
;1718        P 1845  1         CHAR_ENTRY('<', OTHER,  OPCHAR, OPCHAR_INFIX, ADDRESS_OP, TERMINATOR),
;1719        P 1846  1         CHAR_ENTRY('>', OTHER,  OPCHAR, OPCHAR_INFIX, TERMINATOR),
;1720          1847  1         CHAR_ENTRY('=', OTHER,  OPCHAR, OPCHAR_INFIX, TERMINATOR));
;1721          1848  1
;1722          1849  1
;1723          1850  1 ! Define the BASIC Operator Table for operators whose names are identifiers.
;1724          1851  1 !
;1725        P 1852  1 OPERATOR_TABLE(BASIC_IDENT_OPTBL,
;1726        P 1853  1         OPERATOR_ENTRY('NOT',   BIT_NOT,        PREFIX, 200,  45),
;1727        P 1854  1         OPERATOR_ENTRY('AND',   BIT_AND,        INFIX,   40,  40),
;1728        P 1855  1         OPERATOR_ENTRY('OR',    BIT_OR,         INFIX,   30,  30),
;1729        P 1856  1         OPERATOR_ENTRY('XOR',   BIT_XOR,        INFIX,   30,  30),
;1730        P 1857  1         OPERATOR_ENTRY('IMP',   BIT_IMP,        INFIX,   20,  20),
;1731          1858  1         OPERATOR_ENTRY('EQV',   BIT_EQV,        INFIX,   10,  10));
;1732          1859  1
;1733          1860  1
;1734          1861  1 ! Define the BASIC Operator Table for operators whose names are composed of
;1735          1862  1 ! operator characters such as "+", "-", or "*".  This table includes those
;1736          1863  1 ! operators which are part of DEBUG Primary Symbols (such as "\").
;1737          1864  1 !
;1738        P 1865  1 OPERATOR_TABLE(BASIC_OPCHAR_OPTBL,
;1739        P 1866  1         OPERATOR_ENTRY('\',     GLOBAL_SLASH,   PREFIX,   0,   0, PRIMARY),
;1740        P 1867  1         OPERATOR_ENTRY('\',     BACKSLASH,      INFIX,    0,   0, PRIMARY),
;1741        P 1868  1         OPERATOR_ENTRY('::',    DOT,            INFIX,    0,   0, PRIMARY),
;1742        P 1869  1         OPERATOR_ENTRY('(',     SUBSCRIPT,      POSTFIX,  0,   0, PRIMARY),
;1743        P 1870  1
;1744        P 1871  1         OPERATOR_ENTRY('(',     OPENPAREN,      PREFIX, 200,   5, LEXICAL),
;1745        P 1872  1         OPERATOR_ENTRY(')',     CLOSEPAREN,     POSTFIX,  6, 200, LEXICAL),
;1746        P 1873  1         OPERATOR_ENTRY('+',     UNARY_PLUS,     PREFIX, 200,  70),
;1747        P 1874  1         OPERATOR_ENTRY('-',     UNARY_MINUS,    PREFIX, 200,  70),
;1748        P 1875  1         OPERATOR_ENTRY('**',    POWER_OF,       INFIX,   92,  90),
;1749        P 1876  1         OPERATOR_ENTRY('^',     POWER_OF,       INFIX,   92,  90),
;1750        P 1877  1         OPERATOR_ENTRY('*',     MULTIPLY,       INFIX,   80,  80),
;1751        P 1878  1         OPERATOR_ENTRY('/',     DIVIDE,         INFIX,   80,  80),
;1752        P 1879  1         OPERATOR_ENTRY('+',     ADD,            INFIX,   60,  60),
;1753        P 1880  1         OPERATOR_ENTRY('-',     SUBTRACT,       INFIX,   60,  60),
;1754        P 1881  1         OPERATOR_ENTRY('<',     LSS_THAN,       INFIX,   50,  50),
;1755        P 1882  1         OPERATOR_ENTRY('<=',    LSS_EQUAL,      INFIX,   50,  50),
;1756        P 1883  1         OPERATOR_ENTRY('=<',    LSS_EQUAL,      INFIX,   50,  50),
```

```
;  1757        P 1884  1          OPERATOR_ENTRY('>',     GTR_THAN,       INFIX,   50,  50),
;  1758        P 1885  1          OPERATOR_ENTRY('>=',    GTR_EQUAL,      INFIX,   50,  50),
;  1759        P 1886  1          OPERATOR_ENTRY('=>',    GTR_EQUAL,      INFIX,   50,  50),
;  1760        P 1887  1          OPERATOR_ENTRY('=',     EQUAL,          INFIX,   50,  50),
;  1761        P 1888  1          OPERATOR_ENTRY('<>',    NOT_EQUAL,      INFIX,   50,  50),
;  1762          1889  1          OPERATOR_ENTRY('><',    NOT_EQUAL,      INFIX,   50,  50));
;  1763          1890  1
;  1764          1891  1
;  1765          1892  1 ! Define the BASIC Terminator Lexical Token Table for subscript expressions.
;  1766          1893  1 ! In BASIC a subscript expression can be terminated by ")" (end of sub-
;  1767          1894  1 ! scripts), by "," (more subscripts to follow), or by ":" (string subscript
;  1768          1895  1 ! upper bound to follow).
;  1769          1896  1 !
;  1770        P 1897  1 TERMINATOR_TABLE(BASIC_SUBSCR_TERM_TBL,
;  1771        P 1898  1          TERMINATOR_ENTRY(')', TERM_CLOSE, BALANCED_PARENS),
;  1772        P 1899  1          TERMINATOR_ENTRY(':', TERM_COLON, MUST_BE_SINGLE),
;  1773          1900  1          TERMINATOR_ENTRY(',', TERM_COMMA));
;  1774          1901  1
;  1775          1902  1
;  1776          1903  1 ! Define the BASIC Predefined Identifier Table.
;  1777          1904  1 !
;  1778          1905  1 PRID_TABLE(BASIC_PRID_TABLE);
;  1779          1906  1
;  1780          1907  1
;  1781          1908  1 ! Define the BASIC Built-in Function Table.
;  1782          1909  1 !
;  1783          1910  1 BUILT_IN_FUNCTION_TABLE(BASIC_FUNCTION_TABLE);
;  1784          1911  1
;  1785          1912  1
;  1786          1913  1 ! Define the BASIC Number Scanner State Table.  This table defines the states
;  1787          1914  1 ! of a Finite-State Machine which picks up all valid numeric constants in the
;  1788          1915  1 ! language.
;  1789          1916  1 !
;  1790          1917  1 ! The BASIC number table is the same as the UNKNOWN number table at present.
;  1791          1918  1 !
;  1792          1919  1 BIND
;  1793          1920  1     BASIC_NUMBER_TABLE = UNKNOWN_NUMBER_TABLE;
;  1794          1921  1
;  1795          1922  1
;  1796          1923  1 ! Define the Primary Parser State Table for language BASIC Each Transition
;  1797          1924  1 ! Entry in the state table has this format:
;  1798          1925  1 !
;  1799          1926  1 !       PRIMARY_TRANSITION(operator-code, action, next-state)
;  1800          1927  1 !
;  1801          1928  1 ! where the first parameter is the operator code which causes the transition
;  1802          1929  1 ! to be taken, the second parameter is the action routine CASE index for the
;  1803          1930  1 ! transition, and the third parameter is the next state in the Finite-State
;  1804          1931  1 ! Machine.
;  1805          1932  1 !
;  1806        P 1933  1 PRIMARY_STATE_TABLE(BASIC_PRIMARY_TABLE,
;  1807        P 1934  1
;  1808        P 1935  1     PRIMARY_STATE(START_STATE,
;  1809        P 1936  1         PRIMARY_TRANSITION(GLOBAL_SLASH, START_GBL, GET_GLOBAL),
;  1810        P 1937  1         PRIMARY_TRANSITION(BACKSLASH, START_SLASH, GOT_BACKSLASH),
;  1811        P 1938  1         PRIMARY_TRANSITION(INVOCNUM, SLASH_INVOCNUM, GOT_BACKSLASH),
;  1812        P 1939  1         PRIMARY_TRANSITION(DOT, START_DOT, GOT_DOT),
;  1813        P 1940  1         PRIMARY_TRANSITION(SUBSCRIPT, START_SUBSCR, GOT_SUBSCRIPT),
```

```
;  1814        P 1941  1          PRIMARY_TRANSITION(PRIMARY_TERM, START_TERM, END_STATE)),
;  1815        P 1942  1
;  1816        P 1943  1      PRIMARY_STATE(GET_GLOBAL,
;  1817        P 1944  1          PRIMARY_TRANSITION(PRIMARY_TERM, GBL_TERM, END_STATE)),
;  1818        P 1945  1
;  1819        P 1946  1      PRIMARY_STATE(GOT_BACKSLASH,
;  1820        P 1947  1          PRIMARY_TRANSITION(BACKSLASH, SLASH_SLASH, GOT_BACKSLASH),
;  1821        P 1948  1          PRIMARY_TRANSITION(INVOCNUM, SLASH_INVOCNUM, GOT_BACKSLASH),
;  1822        P 1949  1          PRIMARY_TRANSITION(DOT, SLASH_DOT, GOT_DOT),
;  1823        P 1950  1          PRIMARY_TRANSITION(SUBSCRIPT, SLASH_SUBSCR, GOT_SUBSCRIPT),
;  1824        P 1951  1          PRIMARY_TRANSITION(PRIMARY_TERM, SLASH_TERM, END_STATE)),
;  1825        P 1952  1
;  1826        P 1953  1      PRIMARY_STATE(GOT_DOT,
;  1827        P 1954  1          PRIMARY_TRANSITION(DOT, DOT_DOT, GOT_DOT),
;  1828        P 1955  1          PRIMARY_TRANSITION(SUBSCRIPT, DOT_SUBSCR, GOT_SUBSCRIPT),
;  1829        P 1956  1          PRIMARY_TRANSITION(PRIMARY_TERM, DOT_TERM, END_STATE)),
;  1830        P 1957  1
;  1831        P 1958  1      PRIMARY_STATE(GOT_SUBSCRIPT,
;  1832        P 1959  1          PRIMARY_TRANSITION(DOT, SUBSCR_DOT, GOT_DOT),
;  1833        P 1960  1          PRIMARY_TRANSITION(SUBSCRIPT, SUBSCR_SUBSCR, GOT_SUBSCRIPT),
;  1834        P 1961  1          PRIMARY_TRANSITION(PRIMARY_TERM, SUBSCR_TERM, END_STATE)),
;  1835        P 1962  1
;  1836          1963  1      PRIMARY_STATE(END_STATE));
;  1837          1964  1
;  1838          1965  1
;  1839          1966  1  ! Define the table of pointers to the parse tables for BASIC.
;  1840          1967  1  !
;  1841        P 1968  1  LANGUAGE_TABLES(LANGUAGE = BASIC,
;  1842        P 1969  1                  CHARTBL = BASIC_CHARTBL,
;  1843        P 1970  1                  IDENT_OPTBL = BASIC_IDENT_OPTBL,
;  1844        P 1971  1                  OPCHAR_OPTBL = BASIC_OPCHAR_OPTBL,
;  1845        P 1972  1                  NUMBER_TABLE = BASIC_NUMBER_TABLE,
;  1846        P 1973  1                  PRIMARY_TABLE = BASIC_PRIMARY_TABLE,
;  1847        P 1974  1                  SUBSCR_TERMS = BASIC_SUBSCR_TERM_TBL,
;  1848        P 1975  1                  PRIDTBL = BASIC_PRID_TABLE,
;  1849        P 1976  1                  BIF_TABLE = BASIC_FUNCTION_TABLE,
;  1850          1977  1                  INCOMPLETE_QUAL = TRUE);
```

```
;  1852          1978  1 !                          B L I S S   P A R S E   T A B L E S
;  1853          1979  1 !
;  1854          1980  1 !
;  1855          1981  1 !       This sect on includes all the Lexical Scanner and Parser tables needed
;  1856          1982  1 !       to scan and parse the BLISS language.
;  1857          1983  1 !
;  1858          1984  1 !
;  1859          1985  1 !
;  1860          1986  1 !
;  1861          1987  1 ! Define the BLISS Character Table.  What is listed here is actually a list of
;  1862          1988  1 ! exceptions to the Character Table for Language UNKNOWN.
;  1863          1989  1 !
;  1864        P 1990  1 CHAR_EXCEPTION_TABLE(BLISS_CHARTBL,
;  1865        P 1991  1         CHAR_ENTRY('$', OTHER,  IDENT_ANYWHERE),
;  1866        P 1992  1         CHAR_ENTRY('<', OTHER,  OPCHAR, ADDRESS_OP),
;  1867        P 1993  1         CHAR_ENTRY('>', OTHER,  TERMINATOR),
;  1868        P 1994  1         CHAR_ENTRY('[', OTHER,  OPCHAR),
;  1869        P 1995  1         CHAR_ENTRY(']', OTHER,  TERMINATOR),
;  1870        P 1996  1         CHAR_ENTRY('^', OTHER,  OPCHAR, SPECIAL_SYMBOL),
;  1871          1997  1         CHAR_ENTRY('_', OTHER,  IDENT_ANYWHERE));
;  1872          1998  1
;  1873          1999  1 ! Define the BLISS Operator Table for operators whose names are identifiers.
;  1874          2000  1 !
;  1875        P 2001  1 OPERATOR_TABLE(BLISS_IDENT_OPTBL,
;  1876        P 2002  1         OPERATOR_ENTRY('MOD',   REMAINDER,      INFIX,   70,  70),
;  1877        P 2003  1         OPERATOR_ENTRY('EQL',   EQUAL,          INFIX,   50,  50),
;  1878        P 2004  1         OPERATOR_ENTRY('EQLU',  EQUAL,          INFIX,   50,  50),
;  1879        P 2005  1         OPERATOR_ENTRY('EQLA',  EQUAL,          INFIX,   50,  50),
;  1880        P 2006  1         OPERATOR_ENTRY('NEQ',   NOT_EQUAL,      INFIX,   50,  50),
;  1881        P 2007  1         OPERATOR_ENTRY('NEQU',  NOT_EQUAL,      INFIX,   50,  50),
;  1882        P 2008  1         OPERATOR_ENTRY('NEQA',  NOT_EQUAL,      INFIX,   50,  50),
;  1883        P 2009  1         OPERATOR_ENTRY('GTR',   GTR_THAN,       INFIX,   50,  50),
;  1884        P 2010  1         OPERATOR_ENTRY('GTRU',  GTR_THAN_U,     INFIX,   50,  50),
;  1885        P 2011  1         OPERATOR_ENTRY('GTRA',  GTR_THAN_U,     INFIX,   50,  50),
;  1886        P 2012  1         OPERATOR_ENTRY('GEQ',   GTR_EQUAL,      INFIX,   50,  50),
;  1887        P 2013  1         OPERATOR_ENTRY('GEQU',  GTR_EQUAL_U,    INFIX,   50,  50),
;  1888        P 2014  1         OPERATOR_ENTRY('GEQA',  GTR_EQUAL_U,    INFIX,   50,  50),
;  1889        P 2015  1         OPERATOR_ENTRY('LSS',   LSS_THAN,       INFIX,   50,  50),
;  1890        P 2016  1         OPERATOR_ENTRY('LSSU',  LSS_THAN_U,     INFIX,   50,  50),
;  1891        P 2017  1         OPERATOR_ENTRY('LSSA',  LSS_THAN_U,     INFIX,   50,  50),
;  1892        P 2018  1         OPERATOR_ENTRY('LEQ',   LSS_EQUAL,      INFIX,   50,  50),
;  1893        P 2019  1         OPERATOR_ENTRY('LEQU',  LSS_EQUAL_U,    INFIX,   50,  50),
;  1894        P 2020  1         OPERATOR_ENTRY('LEQA',  LSS_EQUAL_U,    INFIX,   50,  50),
;  1895        P 2021  1         OPERATOR_ENTRY('NOT',   BIT_NOT,        PREFIX, 200,  40),
;  1896        P 2022  1         OPERATOR_ENTRY('AND',   BIT_AND,        INFIX,   30,  30),
;  1897        P 2023  1         OPERATOR_ENTRY('OR',    BIT_OR,         INFIX,   20,  20),
;  1898        P 2024  1         OPERATOR_ENTRY('EQV',   BIT_EQV,        INFIX,   10,  10),
;  1899          2025  1         OPERATOR_ENTRY('XOR',   BIT_XOR,        INFIX,   10,  10));
;  1900          2026  1
;  1901          2027  1
;  1902          2028  1 ! Define the BLISS Operator Table for operators whose names are composed of
;  1903          2029  1 ! operator characters such as "+", "-", or "*".  This table includes operators
;  1904          2030  1 ! which are part of DEBUG Primary Symbols (such as "\").
;  1905          2031  1 !
;  1906        P 2032  1 OPERATOR_TABLE(BLISS_OPCHAR_OPTBL,
;  1907        P 2033  1         OPERATOR_ENTRY('\',     GLOBAL_SLASH,   PREFIX, 0, 0, PRIMARY),
;  1908        P 2034  1         OPERATOR_ENTRY('\',     BACKSLASH,      INFIX,  0, 0, PRIMARY),
```

```
;  1909        P 2035  1          OPERATOR_ENTRY('[',     SUBSCRIPT,      POSTFIX,0, 0, PRIMARY),
;  1910        P 2036  1
;  1911        P 2037  1          OPERATOR_ENTRY('(',     OPENPAREN,      PREFIX, 200,   5, LEXICAL),
;  1912        P 2038  1          OPERATOR_ENTRY(')',     CLOSEPAREN,     POSTFIX,  6, 200, LEXICAL),
;  1913        P 2039  1
;  1914        P 2040  1          OPERATOR_ENTRY('<',     BIT_SELECT,     POSTFIX,110, 200, LEXICAL),
;  1915        P 2041  1          OPERATOR_ENTRY('.',     IND_RECT,       PREFIX, 200, 100),
;  1916        P 2042  1          OPERATOR_ENTRY('+',     UNARY_PLUS,     PREFIX, 200,  90),
;  1917        P 2043  1          OPERATOR_ENTRY('-',     UNARY_MINUS,    PREFIX, 200,  90),
;  1918        P 2044  1          OPERATOR_ENTRY('^',     LEFT_SHIFT,     INFIX,   80,  80),
;  1919        P 2045  1          OPERATOR_ENTRY('*',     MULTIPLY,       INFIX,   70,  70),
;  1920        P 2046  1          OPERATOR_ENTRY('/',     DIVIDE,         INFIX,   70,  70),
;  1921        P 2047  1          OPERATOR_ENTRY('+',     ADD,            INFIX,   60,  60),
;  1922          2048  1          OPERATOR_ENTRY('-',     SUBTRACT,       INFIX,   60,  60));
;  1923          2049  1
;  1924          2050  1
;  1925          2051  1  ! Define the BLISS Terminator Lexical Token Table for subscript expressions.
;  1926          2052  1  !
;  1927        P 2053  1  TERMINATOR_TABLE(BLISS_SUBSCR_TERM_TBL,
;  1928        P 2054  1          TERMINATOR_ENTRY(']', TERM_CLOSE),
;  1929        P 2055  1          TERMINATOR_ENTRY(':', TERM_COLON, MUST_BE_SINGLE),
;  1930          2056  1          TERMINATOR_ENTRY(',', TERM_COMMA));
;  1931          2057  1
;  1932          2058  1
;  1933          2059  1  ! Define the BLISS Predefined Identifier Table.
;  1934          2060  1  !
;  1935          2061  1  PRID_TABLE(BLISS_PRID_TABLE);
;  1936          2062  1
;  1937          2063  1
;  1938          2064  1  ! Define the BLISS Built-in Function Table.
;  1939          2065  1  !
;  1940          2066  1  BUILT_IN_FUNCTION_TABLE(BLISS_FUNCTION_TABLE);
;  1941          2067  1
;  1942          2068  1
;  1943          2069  1  ! Define the BLISS Number Scanner State Table.  This table defines the states
;  1944          2070  1  ! of a Finite-State Machine which picks up all valid numeric constants in the
;  1945          2071  1  ! language.
;  1946          2072  1  !
;  1947          2073  1  ! The BLISS number table is the same as the UNKNOWN number table at present.
;  1948          2074  1  !
;  1949          2075  1  BIND
;  1950          2076  1      BLISS_NUMBER_TABLE = UNKNOWN_NUMBER_TABLE;
;  1951          2077  1
;  1952          2078  1
;  1953          2079  1  ! Define the Primary Parser State Table for language BLISS.  Each transition
;  1954          2080  1  ! Entry in the state table has this format:
;  1955          2081  1  !
;  1956          2082  1  !       PRIMARY_TRANSITION(operator-code, action, next-state)
;  1957          2083  1  !
;  1958          2084  1  ! where the first parameter is the operator code which causes the transition
;  1959          2085  1  ! to be taken, the second parameter is the action routine CASE index for the
;  1960          2086  1  ! transition, and the third parameter is the next state in the Finite-State
;  1961          2087  1  ! Machine.
;  1962          2088  1  !
;  1963        P 2089  1  PRIMARY_STATE_TABLE(BLISS_PRIMARY_TABLE,
;  1964        P 2090  1
;  1965        P 2091  1      PRIMARY_STATE(START_STATE,
```

```
;  1966        P 2092  1          PRIMARY_TRANSITION(GLOBAL_SLASH, START_GBL, GET_GLOBAL),
;  1967        P 2093  1          PRIMARY_TRANSITION(BACKSLASH, START_SLASH, GOT_BACKSLASH),
;  1968        P 2094  1          PRIMARY_TRANSITION(INVOCNUM, SLASH_INVOCNUM, GOT_BACKSLASH),
;  1969        P 2095  1          PRIMARY_TRANSITION(SUBSCRIPT, START_SUBSCR_BLI, GOT_SUBSCRIPT),
;  1970        P 2096  1          PRIMARY_TRANSITION(PRIMARY_TERM, START_TERM, END_STATE)),
;  1971        P 2097  1
;  1972        P 2098  1      PRIMARY_STATE(GET_GLOBAL,
;  1973        P 2099  1          PRIMARY_TRANSITION(PRIMARY_TERM, GBL_TERM, END_STATE)),
;  1974        P 2100  1
;  1975        P 2101  1      PRIMARY_STATE(GOT_BACKSLASH,
;  1976        P 2102  1          PRIMARY_TRANSITION(BACKSLASH, SLASH_SLASH, GOT_BACKSLASH),
;  1977        P 2103  1          PRIMARY_TRANSITION(INVOCNUM, SLASH_INVOCNUM, GOT_BACKSLASH),
;  1978        P 2104  1          PRIMARY_TRANSITION(SUBSCRIPT, SLASH_SUBSCR_BLI, GOT_SUBSCRIPT),
;  1979        P 2105  1          PRIMARY_TRANSITION(PRIMARY_TERM, SLASH_TERM, END_STATE)),
;  1980        P 2106  1
;  1981        P 2107  1      PRIMARY_STATE(GOT_SUBSCRIPT,
;  1982        P 2108  1          PRIMARY_TRANSITION(PRIMARY_TERM, SUBSCR_TERM, END_STATE)),
;  1983        P 2109  1
;  1984          2110  1      PRIMARY_STATE(END_STATE));
;  1985          2111  1
;  1986          2112  1
;  1987          2113  1  ! Define the table of pointers to the parse tables for BLISS.
;  1988          2114  1  !
;  1989        P 2115  1  LANGUAGE_TABLES(LANGUAGE = BLISS,
;  1990        P 2116  1                  CHARTBL = BLISS_CHARTBL,
;  1991        P 2117  1                  IDENT_OPTBL = BLISS_IDENT_OPTBL,
;  1992        P 2118  1                  OPCHAR_OPTBL = BLISS_OPCHAR_OPTBL,
;  1993        P 2119  1                  NUMBER_TABLE = BLISS_NUMBER_TABLE,
;  1994        P 2120  1                  PRIMARY_TABLE = BLISS_PRIMARY_TABLE,
;  1995        P 2121  1                  SUBSCR_TERMS = BLISS_SUBSCR_TERM_TBL,
;  1996        P 2122  1                  PRIDTBL = BLISS_PRID_TABLE,
;  1997          2123  1                  BIF_TABLE = BLISS_FUNCTION_TABLE);
```

```
;1999          2124  1 !                          C   P A R S E   T A B L E S
;2000          2125  1 !
;2001          2126  1 !
;2002          2127  1 !
;2003          2128  1 !       This section includes all the Lexical Scanner and Parser tables needed
;2004          2129  1 !       to scan and parse the C language.
;2005          2130  1 !
;2006          2131  1 !
;2007          2132  1 !
;2008          2133  1 ! Define the C Character Table.  What is listed here is actually a list of
;2009          2134  1 ! exceptions to the Character Table for Language UNKNOWN.
;2010          2135  1 !
;2011        P 2136  1 CHAR_EXCEPTION_TABLE(C_CHARTBL,
;2012        P 2137  1         CHAR_ENTRY('$', OTHER,  IDENT_ANYWHERE),
;2013        P 2138  1         CHAR_ENTRY('_', OTHER,  IDENT_ANYWHERE),
;2014        P 2139  1         CHAR_ENTRY('^', OTHER,  OPCHAR, OPCHAR_INFIX),
;2015        P 2140  1         CHAR_ENTRY('*', OTHER,  OPCHAR, ADDRESS_OP),
;2016        P 2141  1         CHAR_ENTRY('%', OTHER,  OPCHAR),
;2017        P 2142  1         CHAR_ENTRY('<', OTHER,  OPCHAR, OPCHAR_INFIX, ADDRESS_OP),
;2018        P 2143  1         CHAR_ENTRY('=', OTHER,  OPCHAR, OPCHAR_INFIX, TERMINATOR),
;2019        P 2144  1         CHAR_ENTRY('>', OTHER,  OPCHAR, OPCHAR_INFIX, TERMINATOR),
;2020        P 2145  1         CHAR_ENTRY('[', OTHER,  OPCHAR),
;2021        P 2146  1         CHAR_ENTRY(']', OTHER,  TERMINATOR),
;2022        P 2147  1         CHAR_ENTRY('^', OTHER,  OPCHAR, SPECIAL_SYMBOL),
;2023        P 2148  1         CHAR_ENTRY('|', OTHER,  OPCHAR, OPCHAR_INFIX),
;2024          2149  1         CHAR_ENTRY('~', OTHER,  OPCHAR));
;2025          2150  1
;2026          2151  1
;2027          2152  1 ! Define the C Operator Table for operators whose names are identifiers.
;2028          2153  1 !
;2029        P 2154  1 OPERATOR_TABLE(C_IDENT_OPTBL,
;2030          2155  1         OPERATOR_ENTRY('SIZEOF', SIZEOF,        PREFIX, 200, 140));
;2031          2156  1
;2032          2157  1
;2033          2158  1 ! Define the C Operator Table for operators whose names are composed of
;2034          2159  1 ! operator characters such as "+", "-", or "*".  This table includes operators
;2035          2160  1 ! which are part of DEBUG Primary Symbols (such as "\").
;2036          2161  1 !
;2037        P 2162  1 OPERATOR_TABLE(C_OPCHAR_OPTBL,
;2038        P 2163  1         OPERATOR_ENTRY('\',     GLOBAL_SLASH,   PREFIX, 0, 0, PRIMARY),
;2039        P 2164  1         OPERATOR_ENTRY('\',     BACKSLASH,      INFIX,  0, 0, PRIMARY),
;2040        P 2165  1         OPERATOR_ENTRY('[',     SUBSCRIPT,      POSTFIX,0, 0, PRIMARY),
;2041        P 2166  1         OPERATOR_ENTRY('.',     DOT,            INFIX,  0, 0, PRIMARY),
;2042        P 2167  1
;2043        P 2168  1         OPERATOR_ENTRY('(',     OPENPAREN,      PREFIX, 200,   5, LEXICAL),
;2044        P 2169  1         OPERATOR_ENTRY(')',     CLOSEPAREN,     POSTFIX,  6, 200, LEXICAL),
;2045        P 2170  1         OPERATOR_ENTRY('!',     NOT,            PREFIX, 200, 140),
;2046        P 2171  1         OPERATOR_ENTRY('~',     BIT_NOT,        PREFIX, 200, 140),
;2047        P 2172  1         OPERATOR_ENTRY('*',     INDIRECT,       PREFIX, 200, 140),
;2048        P 2173  1         OPERATOR_ENTRY('*',     MULTIPLY,       INFIX,  130, 130),
;2049        P 2174  1         OPERATOR_ENTRY('/',     DIVIDE,         INFIX,  130, 130),
;2050        P 2175  1         OPERATOR_ENTRY('%',     REMAINDER,      INFIX,  130, 130),
;2051        P 2176  1         OPERATOR_ENTRY('<<',    LEFT_SHIFT,     INFIX,  110, 110),
;2052        P 2177  1         OPERATOR_ENTRY('>>',    RIGHT_SHIFT,    INFIX,  110, 110),
;2053        P 2178  1         OPERATOR_ENTRY('<',     LSS_THAN,       INFIX,  100, 100),
;2054        P 2179  1         OPERATOR_ENTRY('<=',    LSS_EQUAL,      INFIX,  100, 100),
;2055        P 2180  1         OPERATOR_ENTRY('>',     GTR_THAN,       INFIX,  100, 100),
```

```
;  2056        P 2181  1          OPERATOR_ENTRY('>=',    GTR_EQUAL,      INFIX,  100, 100),
;  2057        P 2182  1          OPERATOR_ENTRY('==',    EQUAL,          INFIX,   90,  90),
;  2058        P 2183  1          OPERATOR_ENTRY('!=',    NOT_EQUAL,      INFIX,   90,  90),
;  2059        P 2184  1          OPERATOR_ENTRY('^',     BIT_XOR,        INFIX,   70,  70),
;  2060        P 2185  1          OPERATOR_ENTRY('|',     BIT_OR,         INFIX,   60,  60),
;  2061          2186  1          OPERATOR_ENTRY('||',    SHORT_OR,       INFIX,   40,  40));
;  2062          2187  1
;  2063          2188  1
;  2064          2189  1  ! Define Lexical Token Entries which require special-case scanning.
;  2065          2190  1  !
;  2066          2191  1  BIND
;  2067          2192  1      C_ADDR_OF_TOKEN =
;  2068          2193  1          OPERATOR_ENTRY('&',     ADDRESS_OF,     PREFIX, 200, 140),
;  2069          2194  1      C_BIT_AND_TOKEN =
;  2070          2195  1          OPERATOR_ENTRY('&',     BIT_AND,        INFIX,   80,  80),
;  2071          2196  1      C_AND_TOKEN =
;  2072          2197  1          OPERATOR_ENTRY('&&',    SHORT_AND,      INFIX,   50,  50),
;  2073          2198  1      C_ADD_TOKEN =
;  2074          2199  1          OPERATOR_ENTRY('+',     ADD,            INFIX,  120, 120),
;  2075          2200  1      C_MINUS_TOKEN =
;  2076          2201  1          OPERATOR_ENTRY('-',     UNARY_MINUS,    PREFIX, 200, 140),
;  2077          2202  1      C_SUB_TOKEN =
;  2078          2203  1          OPERATOR_ENTRY('-',     SUBTRACT,       INFIX,  120, 120),
;  2079          2204  1      C_ARROW_TOKEN =
;  2080          2205  1          OPERATOR_ENTRY('->',    PLI_DEREF,      INFIX,    0,   0, PRIMARY),
;  2081          2206  1
;  2082          2207  1      ! The indirect operator will also be allowed as an address expression
;  2083          2208  1      ! operator in C (synonymous with "." and "@"). That's why it appears
;  2084          2209  1      ! here as a special case, in addition to its appearance in the normal
;  2085          2210  1      ! operator tables. The precedence of "40" here is relative to other
;  2086          2211  1      ! address expression operators.
;  2087          2212  1      !
;  2088          2213  1      C_INDIRECT_TOKEN =
;  2089          2214  1          OPERATOR_ENTRY('*',     INDIRECT,       PREFIX, 200,  40);
;  2090          2215  1
;  2091          2216  1
;  2092          2217  1  ! Increment and decrement were commented out so as not to allow operators with
;  2093          2218  1  ! side effects. If we decide to allow them, the comments can be removed.
;  2094          2219  1  !
;  2095          2220  1  !BIND
;  2096          2221  1  !       C_PRE_INCR_TOKEN =
;  2097          2222  1  !           OPERATOR_ENTRY('++',    PRE_INCR,       PREFIX, 200, 140),
;  2098          2223  1  !       C_POST_INCR_TOKEN =
;  2099          2224  1  !           OPERATOR_ENTRY('++',    POST_INCR,      POSTFIX,140, 200),
;  2100          2225  1  !       C_PRE_DECR_TOKEN =
;  2101          2226  1  !           OPERATOR_ENTRY('--',    PRE_DECR,       PREFIX, 200, 140),
;  2102          2227  1  !       C_POST_DECR_TOKEN =
;  2103          2228  1  !           OPERATOR_ENTRY('--',    POST_DECR,      POSTFIX,140, 200);
;  2104          2229  1
;  2105          2230  1
;  2106          2231  1
;  2107          2232  1  ! Define the C Terminator Lexical Token Table for subscript expressions.
;  2108          2233  1  !
;  2109        P 2234  1  TERMINATOR_TABLE(C_SUBSCR_TERM_TBL,
;  2110        P 2235  1          TERMINATOR_ENTRY(']', TERM_CLOSE),
;  2111        P 2236  1          TERMINATOR_ENTRY(':', TERM_COLON, MUST_BE_SINGLE),
;  2112          2237  1          TERMINATOR_ENTRY(',', TERM_COMMA));
```

```
;  2113          2238  1
;  2114          2239  1
;  2115          2240  1 ! Define the C Predefined Identifier Table.
;  2116          2241  1 !
;  2117          2242  1 PRID_TABLE(C_PRID_TABLE);
;  2118          2243  1
;  2119          2244  1
;  2120          2245  1 ! Define the C Built-in Function Table.
;  2121          2246  1 !
;  2122          2247  1 BUILT_IN_FUNCTION_TABLE(C_FUNCTION_TABLE);
;  2123          2248  1
;  2124          2249  1
;  2125          2250  1 ! Define the C Number Scanner State Table.  This table defines the states
;  2126          2251  1 ! of a Finite-State Machine which picks up all valid numeric constants in the
;  2127          2252  1 ! language.
;  2128          2253  1 !
;  2129          2254  1 ! The C number table is the same as the UNKNOWN number table.
;  2130          2255  1 !
;  2131          2256  1 BIND
;  2132          2257  1     C_NUMBER_TABLE = UNKNOWN_NUMBER_TABLE;
;  2133          2258  1
;  2134          2259  1
;  2135          2260  1 ! Define the Primary Parser State Table for language C.  Each transition
;  2136          2261  1 ! Entry in the state table has this format:
;  2137          2262  1 !
;  2138          2263  1 !       PRIMARY_TRANSITION(operator-code, action, next-state)
;  2139          2264  1 !
;  2140          2265  1 ! where the first parameter is the operator code which causes the transition
;  2141          2266  1 ! to be taken, the second parameter is the action routine CASE index for the
;  2142          2267  1 ! transition, and the third parameter is the next state in the Finite-State
;  2143          2268  1 ! Machine.
;  2144          2269  1 !
;  2145        P 2270  1 PRIMARY_STATE_TABLE(C_PRIMARY_TABLE,
;  2146        P 2271  1
;  2147        P 2272  1     PRIMARY_STATE(START_STATE,
;  2148        P 2273  1         PRIMARY_TRANSITION(GLOBAL_SLASH, START_GBL, GET_GLOBAL),
;  2149        P 2274  1         PRIMARY_TRANSITION(BACKSLASH, START_SLASH, GOT_BACKSLASH),
;  2150        P 2275  1         PRIMARY_TRANSITION(INVOCNUM, SLASH_INVOCNUM, GOT_BACKSLASH),
;  2151        P 2276  1         PRIMARY_TRANSITION(DOT, START_DOT, GOT_DOT),
;  2152        P 2277  1         PRIMARY_TRANSITION(SUBSCRIPT, START_SUBSCR, GOT_SUBSCRIPT),
;  2153        P 2278  1         PRIMARY_TRANSITION(PLI_DEREF, START_DEREF, GOT_DOT),
;  2154        P 2279  1         PRIMARY_TRANSITION(PRIMARY_TERM, START_TERM, END_STATE)),
;  2155        P 2280  1
;  2156        P 2281  1     PRIMARY_STATE(GET_GLOBAL,
;  2157        P 2282  1         PRIMARY_TRANSITION(PRIMARY_TERM, GBL_TERM, END_STATE)),
;  2158        P 2283  1
;  2159        P 2284  1     PRIMARY_STATE(GOT_BACKSLASH,
;  2160        P 2285  1         PRIMARY_TRANSITION(BACKSLASH, SLASH_SLASH, GOT_BACKSLASH),
;  2161        P 2286  1         PRIMARY_TRANSITION(INVOCNUM, SLASH_INVOCNUM, GOT_BACKSLASH),
;  2162        P 2287  1         PRIMARY_TRANSITION(DOT, SLASH_DOT, GOT_DOT),
;  2163        P 2288  1         PRIMARY_TRANSITION(SUBSCRIPT, SLASH_SUBSCR, GOT_SUBSCRIPT),
;  2164        P 2289  1         PRIMARY_TRANSITION(PLI_DEREF, SLASH_DEREF, GOT_DOT),
;  2165        P 2290  1         PRIMARY_TRANSITION(PRIMARY_TERM, SLASH_TERM, END_STATE)),
;  2166        P 2291  1
;  2167        P 2292  1     PRIMARY_STATE(GOT_DOT,
;  2168        P 2293  1         PRIMARY_TRANSITION(DOT, DOT_DOT, GOT_DOT),
;  2169        P 2294  1         PRIMARY_TRANSITION(SUBSCRIPT, DOT_SUBSCR, GOT_SUBSCRIPT),
```

```
;2170        P 2295  1          PRIMARY_TRANSITION(PLI_DEREF, DOT_DEREF, GOT_DOT),
;2171        P 2296  1          PRIMARY_TRANSITION(PRIMARY_TERM, DOT_TERM, END_STATE)),
;2172        P 2297  1
;2173        P 2298  1      PRIMARY_STATE(GOT_SUBSCRIPT,
;2174        P 2299  1          PRIMARY_TRANSITION(DOT, SUBSCR_DOT, GOT_DOT),
;2175        P 2300  1          PRIMARY_TRANSITION(SUBSCRIPT, SUBSCR_SUBSCR, GOT_SUBSCRIPT),
;2176        P 2301  1          PRIMARY_TRANSITION(PLI_DEREF, SUBSCR_DEREF, GOT_DOT),
;2177        P 2302  1          PRIMARY_TRANSITION(PRIMARY_TERM, SUBSCR_TERM, END_STATE)),
;2178        P 2303  1
;2179          2304  1      PRIMARY_STATE(END_STATE));
;2180          2305  1
;2181          2306  1
;2182          2307  1  ! Save away the value of the GOT_SUBSCRIPT state for C. This will be
;2183          2308  1  ! used later in the expression parser.
;2184          2309  1  !
;2185          2310  1  COMPILETIME
;2186          2311  1      REMEMBER_C_STATE_GOT_SUBSCRIPT = 0;
;2187          2312  1  %ASSIGN (REMEMBER_C_STATE_GOT_SUBSCRIPT, PRIMARY$XX_STATE_GOT_SUBSCRIPT);
;2188          2313  1
;2189          2314  1
;2190          2315  1  ! Define the table of pointers to the parse tables for C.
;2191          2316  1  !
;2192        P 2317  1  LANGUAGE_TABLES(LANGUAGE = C,
;2193        P 2318  1                  CHARTBL = C_CHARTBL,
;2194        P 2319  1                  IDENT_OPTBL = C_IDENT_OPTBL,
;2195        P 2320  1                  OPCHAR_OPTBL = C_OPCHAR_OPTBL,
;2196        P 2321  1                  NUMBER_TABLE = C_NUMBER_TABLE,
;2197        P 2322  1                  PRIMARY_TABLE = C_PRIMARY_TABLE,
;2198        P 2323  1                  SUBSCR_TERMS = C_SUBSCR_TERM_TBL,
;2199        P 2324  1                  PRIDTBL = C_PRID_TABLE,
;2200        P 2325  1                  BIF_TABLE = C_FUNCTION_TABLE,
;2201        P 2326  1                  MULTIPLE_SUBSCR = TRUE,
;2202        P 2327  1                  ENFORCE_RECORD = FALSE,
;2203          2328  1                  CASING_SIGNIFICANT = TRUE);
```

```
;2205     2329  1 !                          C O B O L   P A R S E   T A B L E S
;2206     2330  1 !
;2207     2331  1 !
;2208     2332  1 !
;2209     2333  1 !       This section includes all the Lexical Scanner and Parser tables needed
;2210     2334  1 !       to scan and parse the COBOL language.
;2211     2335  1 !
;2212     2336  1 !
;2213     2337  1 ! Define the COBOL Character Table.  What is listed here is actually a list of
;2214     2338  1 ! exceptions to the Character Table for Language UNKNOWN.
;2215     2339  1 !
;2216   P 2340  1 CHAR_EXCEPTION_TABLE(COBOL_CHARTBL,
;2217   P 2341  1         CHAR_ENTRY('-', MINUS,  OPCHAR, ADDRESS_OP, IDENT_MIDDLE),
;2218   P 2342  1         CHAR_ENTRY('0', DIGIT,  DIGIT, IDENT_ANYWHERE, NUMBER_START),
;2219   P 2343  1         CHAR_ENTRY('1', DIGIT,  DIGIT, IDENT_ANYWHERE, NUMBER_START),
;2220   P 2344  1         CHAR_ENTRY('2', DIGIT,  DIGIT, IDENT_ANYWHERE, NUMBER_START),
;2221   P 2345  1         CHAR_ENTRY('3', DIGIT,  DIGIT, IDENT_ANYWHERE, NUMBER_START),
;2222   P 2346  1         CHAR_ENTRY('4', DIGIT,  DIGIT, IDENT_ANYWHERE, NUMBER_START),
;2223   P 2347  1         CHAR_ENTRY('5', DIGIT,  DIGIT, IDENT_ANYWHERE, NUMBER_START),
;2224   P 2348  1         CHAR_ENTRY('6', DIGIT,  DIGIT, IDENT_ANYWHERE, NUMBER_START),
;2225   P 2349  1         CHAR_ENTRY('7', DIGIT,  DIGIT, IDENT_ANYWHERE, NUMBER_START),
;2226   P 2350  1         CHAR_ENTRY('8', DIGIT,  DIGIT, IDENT_ANYWHERE, NUMBER_START),
;2227   P 2351  1         CHAR_ENTRY('9', DIGIT,  DIGIT, IDENT_ANYWHERE, NUMBER_START),
;2228   P 2352  1         CHAR_ENTRY('<', OTHER,  OPCHAR, OPCHAR_INFIX, ADDRESS_OP, TERMINATOR),
;2229   P 2353  1         CHAR_ENTRY('>', OTHER,  OPCHAR, OPCHAR_INFIX, TERMINATOR),
;2230   P 2354  1         CHAR_ENTRY('=', OTHER,  OPCHAR, OPCHAR_INFIX, TERMINATOR),
;2231   P 2355  1         CHAR_ENTRY('_', OTHER,  IDENT_ANYWHERE),
;2232     2356  1         CHAR_ENTRY('$', OTHER,  IDENT_ANYWHERE));
;2233     2357  1
;2234     2358  1
;2235     2359  1 ! Define the COBOL Operator Table for operators whose names are identifiers.
;2236     2360  1 !
;2237   P 2361  1 OPERATOR_TABLE(COBOL_IDENT_OPTBL,
;2238   P 2362  1         OPERATOR_ENTRY('OF',    DOT,            INFIX,  0, 0, PRIMARY),
;2239   P 2363  1         OPERATOR_ENTRY('IN',    DOT,            INFIX,  0, 0, PRIMARY),
;2240   P 2364  1         OPERATOR_ENTRY('NOT',   NOT,            PREFIX, 200, 25),
;2241   P 2365  1         OPERATOR_ENTRY('AND',   AND,            INFIX,   20, 20),
;2242   P 2366  1         OPERATOR_ENTRY('OR',    OR,             INFIX,   10, 10),
;2243     2367  1         OPERATOR_ENTRY('NOT',   INFIX_NOT,      INFIX,   30, 30, LEXICAL));
;2244     2368  1
;2245     2369  1 BIND
;2246     2370  1         COBOL_NOT_EQL_TOKEN =
;2247     2371  1             OPERATOR_ENTRY('NOT =', NOT_EQUAL,      INFIX,   30, 30),
;2248     2372  1         COBOL_NOT_GTR_TOKEN =
;2249     2373  1             OPERATOR_ENTRY('NOT >', LSS_EQUAL,      INFIX,   30, 30),
;2250     2374  1         COBOL_NOT_LSS_TOKEN =
;2251     2375  1             OPERATOR_ENTRY('NOT <', GTR_EQUAL,      INFIX,   30, 30);
;2252     2376  1
;2253     2377  1
;2254     2378  1 ! Define the COBOL Operator Table for operators whose names are composed of
;2255     2379  1 ! operator characters such as "+", "-", or "*".  This table includes operators
;2256     2380  1 ! which are part of DEBUG Primary Symbols (such as "\").
;2257     2381  1 !
;2258   P 2382  1 OPERATOR_TABLE(COBOL_OPCHAR_OPTBL,
;2259   P 2383  1         OPERATOR_ENTRY('\',     GLOBAL_SLASH,   PREFIX, 0, 0, PRIMARY),
;2260   P 2384  1         OPERATOR_ENTRY('\',     BACKSLASH,      INFIX,  0, 0, PRIMARY),
;2261   P 2385  1         OPERATOR_ENTRY('(',     SUBSCRIPT,      POSTFIX,0, 0, PRIMARY),
```

```
;  2262        P 2386  1          OPERATOR_ENTRY('>',     PREFIX_GTR,     PREFIX, 200,  30, LEXICAL),
;  2263        P 2387  1          OPERATOR_ENTRY('<',     PREFIX_LSS,     PREFIX, 200,  30, LEXICAL),
;  2264        P 2388  1          OPERATOR_ENTRY('=',     PREFIX_EQL,     PREFIX, 200,  30, LEXICAL),
;  2265        P 2389  1          OPERATOR_ENTRY('(',     OPENPAREN,      PREFIX, 200,   5, LEXICAL),
;  2266        P 2390  1          OPERATOR_ENTRY(')',     CLOSEPAREN,     POSTFIX,  6, 200, LEXICAL),
;  2267        P 2391  1  !       OPERATOR_ENTRY('**',    POWER_OF,       INFIX,   70,  70),
;  2268        P 2392  1          OPERATOR_ENTRY('*',     MULTIPLY,       INFIX,   60,  60),
;  2269        P 2393  1          OPERATOR_ENTRY('/',     DIVIDE,         INFIX,   60,  60),
;  2270        P 2394  1          OPERATOR_ENTRY('+',     UNARY_PLUS,     PREFIX, 200,  50),
;  2271        P 2395  1          OPERATOR_ENTRY('-',     UNARY_MINUS,    PREFIX, 200,  50),
;  2272        P 2396  1          OPERATOR_ENTRY('+',     ADD,            INFIX,   40,  40),
;  2273        P 2397  1          OPERATOR_ENTRY('-',     SUBTRACT,       INFIX,   40,  40),
;  2274        P 2398  1          OPERATOR_ENTRY('>',     GTR_THAN,       INFIX,   30,  30),
;  2275        P 2399  1          OPERATOR_ENTRY('<',     LSS_THAN,       INFIX,   30,  30),
;  2276          2400  1          OPERATOR_ENTRY('=',     EQUAL,          INFIX,   30,  30));
;  2277          2401  1
;  2278          2402  1
;  2279          2403  1  ! Define the COBOL Terminator Lexical Token Table for subscript expressions.
;  2280          2404  1  !
;  2281        P 2405  1  TERMINATOR_TABLE(COBOL_SUBSCR_TERM_TBL,
;  2282        P 2406  1          TERMINATOR_ENTRY(')', TERM_CLOSE),
;  2283        P 2407  1          TERMINATOR_ENTRY(':', TERM_COLON, MUST_BE_SINGLE),
;  2284          2408  1          TERMINATOR_ENTRY(',', TERM_COMMA));
;  2285          2409  1
;  2286          2410  1
;  2287          2411  1  ! Define the COBOL Predefined Identifier Table.
;  2288          2412  1  !
;  2289          2413  1  PRID_TABLE(COBOL_PRID_TABLE);
;  2290          2414  1
;  2291          2415  1
;  2292          2416  1  ! Define the COBOL Built-in Function Table.
;  2293          2417  1  !
;  2294          2418  1  BUILT_IN_FUNCTION_TABLE(COBOL_FUNCTION_TABLE);
;  2295          2419  1
;  2296          2420  1
;  2297          2421  1  ! Define the COBOL Number Scanner State Table.  This table defines the states
;  2298          2422  1  ! of a Finite-State Machine which picks up all valid numeric constants in the
;  2299          2423  1  ! language.  Each Transition Entry is of the form:
;  2300          2424  1  !
;  2301          2425  1  !       NUMBER_TRANSITION(character-class, action-index, next-state)
;  2302          2426  1  !
;  2303          2427  1  ! where the character-class and action-index names are automatically prefixed
;  2304          2428  1  ! by 'NUMST$K_CLASS_' or 'NUMST$K_ACT_' by the NUMBER_TRANSITION macro.
;  2305          2429  1  !
;  2306        P 2430  1  NUMBER_STATE_TABLE(COBOL_NUMBER_TABLE,
;  2307        P 2431  1
;  2308        P 2432  1      NUMBER_STATE(START_STATE,
;  2309        P 2433  1          NUMBER_TRANSITION(DIGIT, GO_PAST_PACK, ACCUM_INT),
;  2310        P 2434  1          NUMBER_TRANSITION(DOT, DO_NOTHING, LEADING_DOT),
;  2311        P 2435  1          NUMBER_TRANSITION(OTHER, NOT_NUMBER, END_STATE)),
;  2312        P 2436  1
;  2313        P 2437  1      NUMBER_STATE(LEADING_DOT,
;  2314        P 2438  1          NUMBER_TRANSITION(DIGIT, GO_PAST_PACK_FRAC, ACCUM_FRAC),
;  2315        P 2439  1          NUMBER_TRANSITION(OTHER, NOT_NUMBER, END_STATE)),
;  2316        P 2440  1
;  2317        P 2441  1      NUMBER_STATE(ACCUM_INT,
;  2318        P 2442  1          NUMBER_TRANSITION(DIGIT, GO_PAST_PACK, ACCUM_INT),
```

```
;  2319        P 2443  1          NUMBER_TRANSITION(DOT, DO_NOTHING, ACCUM_FRAC),
;  2320        P 2444  1          NUMBER_TRANSITION(HEXDIGIT, COB_CKHEX, ACCUM_HEX),
;  2321        P 2445  1          NUMBER_TRANSITION(B, COB_CKHEX, ACCUM_HEX),
;  2322        P 2446  1          NUMBER_TRANSITION(D, COB_CKHEX, ACCUM_HEX),
;  2323        P 2447  1          NUMBER_TRANSITION(E, COB_CKHEX, ACCUM_HEX),
;  2324        P 2448  1          NUMBER_TRANSITION(OTHER, COB_CKNUM, END_STATE)),
;  2325        P 2449  1
;  2326        P 2450  1      NUMBER_STATE(ACCUM_HEX,
;  2327        P 2451  1          NUMBER_TRANSITION(DIGIT, DO_NOTHING, ACCUM_HEX),
;  2328        P 2452  1          NUMBER_TRANSITION(HEXDIGIT, DO_NOTHING, ACCUM_HEX),
;  2329        P 2453  1          NUMBER_TRANSITION(B, DO_NOTHING, ACCUM_HEX),
;  2330        P 2454  1          NUMBER_TRANSITION(D, DO_NOTHING, ACCUM_HEX),
;  2331        P 2455  1          NUMBER_TRANSITION(E, DO_NOTHING, ACCUM_HEX),
;  2332        P 2456  1          NUMBER_TRANSITION(OTHER, COB_CKNUM, END_STATE)),
;  2333        P 2457  1
;  2334        P 2458  1      NUMBER_STATE(ACCUM_FRAC,
;  2335        P 2459  1          NUMBER_TRANSITION(DIGIT, GO_PAST_PACK_FRAC, ACCUM_FRAC),
;  2336        P 2460  1          NUMBER_TRANSITION(DOT, BACKUP_PTRS, END_STATE),
;  2337        P 2461  1          NUMBER_TRANSITION(E, MARK_E_EXP, GET_EXPONENT),
;  2338        P 2462  1          NUMBER_TRANSITION(D, MARK_D_EXP, GET_EXPONENT),
;  2339        P 2463  1          NUMBER_TRANSITION(G, MARK_G_EXP, GET_EXPONENT),
;  2340        P 2464  1          NUMBER_TRANSITION(Q, MARK_Q_EXP, GET_EXPONENT),
;  2341        P 2465  1          NUMBER_TRANSITION(OTHER, GOT_PACK_NUMBER, END_STATE)),
;  2342        P 2466  1
;  2343        P 2467  1      NUMBER_STATE(GET_EXPONENT,
;  2344        P 2468  1          NUMBER_TRANSITION(DIGIT, DO_NOTHING, ACCUM_EXP),
;  2345        P 2469  1          NUMBER_TRANSITION(PLUS, DO_NOTHING, GET_EXP_SIGN),
;  2346        P 2470  1          NUMBER_TRANSITION(MINUS, DO_NOTHING, GET_EXP_SIGN),
;  2347        P 2471  1          NUMBER_TRANSITION(OTHER, BACKUP_PTRS, END_STATE)),
;  2348        P 2472  1
;  2349        P 2473  1      NUMBER_STATE(GET_EXP_SIGN,
;  2350        P 2474  1          NUMBER_TRANSITION(DIGIT, DO_NOTHING, ACCUM_EXP),
;  2351        P 2475  1          NUMBER_TRANSITION(OTHER, GOT_NUMBER, END_STATE)),
;  2352        P 2476  1
;  2353        P 2477  1      NUMBER_STATE(ACCUM_EXP,
;  2354        P 2478  1          NUMBER_TRANSITION(DIGIT, DO_NOTHING, ACCUM_EXP),
;  2355        P 2479  1          NUMBER_TRANSITION(OTHER, GOT_NUMBER, END_STATE)),
;  2356        P 2480  1
;  2357        P 2481  1      NUMBER_STATE(END_STATE,
;  2358          2482  1          NUMBER_TRANSITION(OTHER, GIVE_ERROR, END_STATE)));
;  2359          2483  1
;  2360          2484  1
;  2361          2485  1  ! Define the Primary Parser State Table for language COBOL.  Each transition
;  2362          2486  1  ! Entry in the state table has this format:
;  2363          2487  1  !
;  2364          2488  1  !       PRIMARY_TRANSITION(operator-code, action, next-state)
;  2365          2489  1  !
;  2366          2490  1  ! where the first parameter is the operator code which causes the transition
;  2367          2491  1  ! to be taken, the second parameter is the action routine CASE index for the
;  2368          2492  1  ! transition, and the third parameter is the next state in the Finite-State
;  2369          2493  1  ! Machine.
;  2370          2494  1  !
;  2371        P 2495  1  PRIMARY_STATE_TABLE(COBOL_PRIMARY_TABLE,
;  2372        P 2496  1
;  2373        P 2497  1      PRIMARY_STATE(START_STATE,
;  2374        P 2498  1          PRIMARY_TRANSITION(GLOBAL_SLASH, START_GBL, GET_GLOBAL),
;  2375        P 2499  1          PRIMARY_TRANSITION(BACKSLASH, START_SLASH, GOT_BACKSLASH),
```

```
;  2376        P 2500  1          PRIMARY_TRANSITION(INVOCNUM, SLASH_INVOCNUM, GOT_BACKSLASH),
;  2377        P 2501  1          PRIMARY_TRANSITION(DOT, START_DOT_COB, GOT_DOT),
;  2378        P 2502  1          PRIMARY_TRANSITION(SUBSCRIPT, START_SUBSCR_PLI, GOT_SUBSCRIPT),
;  2379        P 2503  1          PRIMARY_TRANSITION(PRIMARY_TERM, START_TERM, END_STATE)),
;  2380        P 2504  1
;  2381        P 2505  1      PRIMARY_STATE(GET_GLOBAL,
;  2382        P 2506  1          PRIMARY_TRANSITION(PRIMARY_TERM, GBL_TERM, END_STATE)),
;  2383        P 2507  1
;  2384        P 2508  1      PRIMARY_STATE(GOT_BACKSLASH,
;  2385        P 2509  1          PRIMARY_TRANSITION(BACKSLASH, SLASH_SLASH, GOT_BACKSLASH),
;  2386        P 2510  1          PRIMARY_TRANSITION(INVOCNUM, SLASH_INVOCNUM, GOT_BACKSLASH),
;  2387        P 2511  1          PRIMARY_TRANSITION(SUBSCRIPT, SLASH_SUBSCR_PLI, GOT_SUBSCRIPT),
;  2388        P 2512  1          PRIMARY_TRANSITION(PRIMARY_TERM, SLASH_TERM, END_STATE)),
;  2389        P 2513  1
;  2390        P 2514  1      PRIMARY_STATE(GOT_DOT,
;  2391        P 2515  1          PRIMARY_TRANSITION(BACKSLASH, DOT_SLASH_COB, GOT_BACKSLASH),
;  2392        P 2516  1          PRIMARY_TRANSITION(DOT, DOT_DOT_COB, GOT_DOT),
;  2393        P 2517  1          PRIMARY_TRANSITION(SUBSCRIPT, DOT_SUBSCR_COB, GOT_SUBSCRIPT),
;  2394        P 2518  1          PRIMARY_TRANSITION(PRIMARY_TERM, DOT_TERM_COB, END_STATE)),
;  2395        P 2519  1
;  2396        P 2520  1      PRIMARY_STATE(GOT_SUBSCRIPT,
;  2397        P 2521  1          PRIMARY_TRANSITION(SUBSCRIPT, SUBSCR_SUBSCR_PLI, GOT_SUBSCRIPT2),
;  2398        P 2522  1          PRIMARY_TRANSITION(PRIMARY_TERM, SUBSCR_TERM_PLI, END_STATE)),
;  2399        P 2523  1
;  2400        P 2524  1      PRIMARY_STATE(GOT_SUBSCRIPT2,
;  2401        P 2525  1          PRIMARY_TRANSITION(PRIMARY_TERM, SUBSCR_TERM_PLI, END_STATE)),
;  2402        P 2526  1
;  2403          2527  1      PRIMARY_STATE(END_STATE));
;  2404          2528  1
;  2405          2529  1
;  2406          2530  1  ! Define the table of pointers to the parse tables for COBOL.
;  2407          2531  1  !
;  2408        P 2532  1  LANGUAGE_TABLES(LANGUAGE = COBOL,
;  2409        P 2533  1                  CHARTBL = COBOL_CHARTBL,
;  2410        P 2534  1                  IDENT_OPTBL = COBOL_IDENT_OPTBL,
;  2411        P 2535  1                  OPCHAR_OPTBL = COBOL_OPCHAR_OPTBL,
;  2412        P 2536  1                  NUMBER_TABLE = COBOL_NUMBER_TABLE,
;  2413        P 2537  1                  PRIMARY_TABLE = COBOL_PRIMARY_TABLE,
;  2414        P 2538  1                  SUBSCR_TERMS = COBOL_SUBSCR_TERM_TBL,
;  2415        P 2539  1                  PRIDTBL = COBOL_PRID_TABLE,
;  2416        P 2540  1                  BIF_TABLE = COBOL_FUNCTION_TABLE,
;  2417        P 2541  1                  MULTIPLE_SUBSCR = FALSE,
;  2418          2542  1                  COMPONENTS_IN_PATHNAME = TRUE);
```

```
;2420          2543  1 !                          F O R T R A N   P A R S E   T A B L E S
;2421          2544  1 !
;2422          2545  1 !
;2423          2546  1 !
;2424          2547  1 !       This section includes all the Lexical Scanner and Parser tables needed
;2425          2548  1 !       to scan and parse FORTRAN expressions.
;2426          2549  1 !
;2427          2550  1 !
;2428          2551  1 !
;2429          2552  1 ! Define the FORTRAN Character Table.  What is listed here is actually a list
;2430          2553  1 ! of exceptions to the Character Table for language UNKNOWN.  (Language UNKNOWN
;2431          2554  1 ! lists the "average" use of each character in the character set.)
;2432          2555  1 !
;2433        P 2556  1 CHAR_EXCEPTION_TABLE(FORTRAN_CHARTBL,
;2434        P 2557  1         CHAR_ENTRY('.', DOT,    NUMBER_START, SPECIAL_SYMBOL),
;2435          2558  1         CHAR_ENTRY('/', OTHER,  OPCHAR, OPCHAR_INFIX));
;2436          2559  1
;2437          2560  1
;2438          2561  1 ! Define the FORTRAN Operator Table for operators whose names are identifiers.
;2439          2562  1 ! This table is empty since FORTRAN has no such operators, but the Lexical
;2440          2563  1 ! Scanner requires that such a table exist anyway.
;2441          2564  1 !
;2442          2565  1 OPERATOR_TABLE(FORTRAN_IDENT_OPTBL);
;2443          2566  1
;2444          2567  1
;2445          2568  1 ! Define the FORTRAN Operator Table for operators whose names are composed of
;2446          2569  1 ! operator characters such as "+", "-", or "*".  This table includes those
;2447          2570  1 ! operators which are part of DEBUG Primary Symbols (such as "\").
;2448          2571  1 !
;2449        P 2572  1 OPERATOR_TABLE(FORTRAN_OPCHAR_OPTBL,
;2450        P 2573  1         OPERATOR_ENTRY('\',     GLOBAL_SLASH,   PREFIX,   0,   0, PRIMARY),
;2451        P 2574  1         OPERATOR_ENTRY('\',     BACKSLASH,      INFIX,    0,   0, PRIMARY),
;2452        P 2575  1         OPERATOR_ENTRY('(',     SUBSCRIPT,      POSTFIX,  0,   0, PRIMARY),
;2453        P 2576  1         OPERATOR_ENTRY('.',     DOT,            INFIX,    0,   0, PRIMARY),
;2454        P 2577  1         OPERATOR_ENTRY('+',     ADD,            INFIX,   60,  60),
;2455        P 2578  1         OPERATOR_ENTRY('-',     SUBTRACT,       INFIX,   60,  60),
;2456        P 2579  1         OPERATOR_ENTRY('+',     UNARY_PLUS,     PREFIX, 200,  70),
;2457        P 2580  1         OPERATOR_ENTRY('-',     UNARY_MINUS,    PREFIX, 200,  70),
;2458        P 2581  1         OPERATOR_ENTRY('*',     MULTIPLY,       INFIX,   80,  80),
;2459        P 2582  1         OPERATOR_ENTRY('/',     DIVIDE,         INFIX,   80,  80),
;2460        P 2583  1         OPERATOR_ENTRY('**',    POWER_OF,       INFIX,   92,  90),
;2461        P 2584  1         OPERATOR_ENTRY('//',    CONCATENATE,    INFIX,   60,  60),
;2462        P 2585  1         OPERATOR_ENTRY('(',     OPENPAREN,      PREFIX, 200,   5, LEXICAL),
;2463          2586  1         OPERATOR_ENTRY(')',     CLOSEPAREN,     POSTFIX,  6, 200, LEXICAL));
;2464          2587  1
;2465          2588  1
;2466          2589  1 BIND
;2467          2590  1         FORTRAN_INDIRECT_TOKEN = OPERATOR_ENTRY('.',INDIRECT, PREFIX, 200, 40),
;2468          2591  1         FORTRAN_DOT_TOKEN = OPERATOR_ENTRY('.', DOT, INFIX, 0, 0, PRIMARY);
;2469          2592  1
;2470          2593  1
;2471          2594  1 ! Define an Operator Table for FORTRAN operators of the form .XX. or .XXX.
;2472          2595  1 ! This includes all the FORTRAN comparison and boolean operators.
;2473          2596  1 !
;2474        P 2597  1 OPERATOR_TABLE(FORTRAN_SPECIAL_OPTBL,
;2475        P 2598  1         OPERATOR_ENTRY('.EQ.',  EQUAL,          INFIX,   50,  50),
;2476        P 2599  1         OPERATOR_ENTRY('.NE.',  NOT_EQUAL,      INFIX,   50,  50),
```

```
;  2477        P 2600  1          OPERATOR_ENTRY('.GT.',  GTR_THAN,        INFIX,   50, 50),
;  2478        P 2601  1          OPERATOR_ENTRY('.GE.',  GTR_EQUAL,       INFIX,   50, 50),
;  2479        P 2602  1          OPERATOR_ENTRY('.LT.',  LSS_THAN,        INFIX,   50, 50),
;  2480        P 2603  1          OPERATOR_ENTRY('.LE.',  LSS_EQUAL,       INFIX,   50, 50),
;  2481        P 2604  1          OPERATOR_ENTRY('.NOT.', NOT,             PREFIX, 200, 40),
;  2482        P 2605  1          OPERATOR_ENTRY('.AND.', AND,             INFIX,   30, 30),
;  2483        P 2606  1          OPERATOR_ENTRY('.OR.',  OR,              INFIX,   20, 20),
;  2484        P 2607  1          OPERATOR_ENTRY('.XOR.', XOR,             INFIX,   10, 10),
;  2485        P 2608  1          OPERATOR_ENTRY('.EQV.', EQV,             INFIX,   10, 10),
;  2486          2609  1          OPERATOR_ENTRY('.NEQV.',XOR,             INFIX,   10, 10));
;  2487          2610  1
;  2488          2611  1
;  2489          2612  1  ! Define the FORTRAN Terminator Lexical Token Table for subscript expressions.
;  2490          2613  1  ! In FORTRAN, a subscript expression can be terminated by ")" (end of sub-
;  2491          2614  1  ! scripts), by "," (more subscripts to follow), or by ":" (string subscript
;  2492          2615  1  ! upper bound to follow).
;  2493          2616  1  !
;  2494        P 2617  1  TERMINATOR_TABLE(FORTRAN_SUBSCR_TERM_TBL,
;  2495        P 2618  1          TERMINATOR_ENTRY(')', TERM_CLOSE, BALANCED_PARENS),
;  2496        P 2619  1          TERMINATOR_ENTRY(',', TERM_COMMA),
;  2497          2620  1          TERMINATOR_ENTRY(':', TERM_COLON));
;  2498          2621  1
;  2499          2622  1
;  2500          2623  1  ! Define the FORTRAN Predefined Identifier Table.
;  2501          2624  1  !
;  2502        P 2625  1  PRID_TABLE(FORTRAN_PRID_TABLE,
;  2503        P 2626  1          PRID_ENTRY('.TRUE.', ATOMIC, L, 1),
;  2504          2627  1          PRID_ENTRY('.FALSE.', ATOMIC, L, 0));
;  2505          2628  1
;  2506          2629  1
;  2507          2630  1  ! Define the FORTRAN Built-in Function Table.
;  2508          2631  1  !
;  2509          2632  1  BUILT_IN_FUNCTION_TABLE(FORTRAN_FUNCTION_TABLE);
;  2510          2633  1
;  2511          2634  1
;  2512          2635  1  ! Define the FORTRAN Number Scanner State Table.  This table defines the states
;  2513          2636  1  ! of a Finite-State Machine which picks up all valid numeric constants in the
;  2514          2637  1  ! language.
;  2515          2638  1  !
;  2516          2639  1  ! The FORTRAN number table is the same as the number table for language UNKNOWN.
;  2517          2640  1  !
;  2518          2641  1  BIND
;  2519          2642  1      FORTRAN_NUMBER_TABLE = UNKNOWN_NUMBER_TABLE;
;  2520          2643  1
;  2521          2644  1
;  2522          2645  1  ! Define the Primary Parser State Table for language FORTRAN.  Each Transition
;  2523          2646  1  ! Entry in the state table has this format:
;  2524          2647  1  !
;  2525          2648  1  !       PRIMARY_TRANSITION(operator-code, action, next-state)
;  2526          2649  1  !
;  2527          2650  1  ! where the first parameter is the operator code which causes the transition
;  2528          2651  1  ! to be taken, the second parameter is the action routine CASE index for the
;  2529          2652  1  ! transition, and the third parameter is the next state in the Finite-State
;  2530          2653  1  ! Machine.
;  2531          2654  1  !
;  2532        P 2655  1  PRIMARY_STATE_TABLE(FORTRAN_PRIMARY_TABLE,
;  2533        P 2656  1
```

```
;  2534        P 2657  1      PRIMARY_STATE(START_STATE,
;  2535        P 2658  1          PRIMARY_TRANSITION(GLOBAL_SLASH, START_GBL, GET_GLOBAL),
;  2536        P 2659  1          PRIMARY_TRANSITION(BACKSLASH, START_SLASH, GOT_BACKSLASH),
;  2537        P 2660  1          PRIMARY_TRANSITION(INVOCNUM, SLASH_INVOCNUM, GOT_BACKSLASH),
;  2538        P 2661  1          PRIMARY_TRANSITION(DOT, START_DOT, GOT_DOT),
;  2539        P 2662  1          PRIMARY_TRANSITION(SUBSCRIPT, START_SUBSCR, GOT_SUBSCRIPT),
;  2540        P 2663  1          PRIMARY_TRANSITION(PRIMARY_TERM, START_TERM, END_STATE)),
;  2541        P 2664  1
;  2542        P 2665  1      PRIMARY_STATE(GET_GLOBAL,
;  2543        P 2666  1          PRIMARY_TRANSITION(PRIMARY_TERM, GBL_TERM, END_STATE)),
;  2544        P 2667  1
;  2545        P 2668  1      PRIMARY_STATE(GOT_BACKSLASH,
;  2546        P 2669  1          PRIMARY_TRANSITION(BACKSLASH, SLASH_SLASH, GOT_BACKSLASH),
;  2547        P 2670  1          PRIMARY_TRANSITION(INVOCNUM, SLASH_INVOCNUM, GOT_BACKSLASH),
;  2548        P 2671  1          PRIMARY_TRANSITION(SUBSCRIPT, SLASH_SUBSCR, GOT_SUBSCRIPT),
;  2549        P 2672  1          PRIMARY_TRANSITION(PRIMARY_TERM, SLASH_TERM, END_STATE)),
;  2550        P 2673  1
;  2551        P 2674  1      PRIMARY_STATE(GOT_DOT,
;  2552        P 2675  1          PRIMARY_TRANSITION(DOT, DOT_DOT, GOT_DOT),
;  2553        P 2676  1          PRIMARY_TRANSITION(SUBSCRIPT, DOT_SUBSCR, GOT_SUBSCRIPT),
;  2554        P 2677  1          PRIMARY_TRANSITION(PRIMARY_TERM, DOT_TERM, END_STATE)),
;  2555        P 2678  1
;  2556        P 2679  1      PRIMARY_STATE(GOT_SUBSCRIPT,
;  2557        P 2680  1          PRIMARY_TRANSITION(DOT, SUBSCR_DOT, GOT_DOT),
;  2558        P 2681  1          PRIMARY_TRANSITION(SUBSCRIPT, SUBSCR_SUBSCR, GOT_SUBSCRIPT),
;  2559        P 2682  1          PRIMARY_TRANSITION(PRIMARY_TERM, SUBSCR_TERM, END_STATE)),
;  2560        P 2683  1
;  2561          2684  1      PRIMARY_STATE(END_STATE));
;  2562          2685  1
;  2563          2686  1
;  2564          2687  1  ! Define the table of pointers to the parse tables for FORTRAN.
;  2565          2688  1  !
;  2566        P 2689  1  LANGUAGE_TABLES(LANGUAGE = FORTRAN,
;  2567        P 2690  1                  CHARTBL = FORTRAN_CHARTBL,
;  2568        P 2691  1                  IDENT_OPTBL = FORTRAN_IDENT_OPTBL,
;  2569        P 2692  1                  OPCHAR_OPTBL = FORTRAN_OPCHAR_OPTBL,
;  2570        P 2693  1                  NUMBER_TABLE = FORTRAN_NUMBER_TABLE,
;  2571        P 2694  1                  PRIMARY_TABLE = FORTRAN_PRIMARY_TABLE,
;  2572        P 2695  1                  SUBSCR_TERMS = FORTRAN_SUBSCR_TERM_TBL,
;  2573        P 2696  1                  PRIDTBL = FORTRAN_PRID_TABLE,
;  2574          2697  1                  BIF_TABLE = FORTRAN_FUNCTION_TABLE);
```

```
;  2576          2698  1 !                          M A C R O   P A R S E   T A B L E S
;  2577          2699  1 !
;  2578          2700  1 !
;  2579          2701  1 !
;  2580          2702  1 !       This section includes all the Lexical Scanner and Parser tables needed
;  2581          2703  1 !       to scan and parse the MACRO language.
;  2582          2704  1 !
;  2583          2705  1 !
;  2584          2706  1 !
;  2585          2707  1 ! Define the MACRO Character Table.  What is listed here is actually a list of
;  2586          2708  1 ! exceptions to the Character Table for Language UNKNOWN.
;  2587          2709  1 !
;  2588        P 2710  1 CHAR_EXCEPTION_TABLE(MACRO_CHARTBL,
;  2589        P 2711  1         CHAR_ENTRY(' ', OTHER, IDENT_ANYWHERE),
;  2590        P 2712  1         CHAR_ENTRY('$', OTHER, IDENT_ANYWHERE),
;  2591        P 2713  1         CHAR_ENTRY('.', DOT,   NUMBER_START, OPCHAR, ADDRESS_OP, SPECIAL_SYMBOL,
;  2592        P 2714  1                                IDENT_MIDDLE, IDENT_END),
;  2593        P 2715  1         CHAR_ENTRY('<', OTHER, OPCHAR, ADDRESS_OP),
;  2594          2716  1         CHAR_ENTRY('>', OTHER, TERMINATOR));
;  2595          2717  1
;  2596          2718  1
;  2597          2719  1 ! Define the MACRO Operator Table for operators whose names are identifiers.
;  2598          2720  1 !
;  2599        P 2721  1 OPERATOR_TABLE(MACRO_IDENT_OPTBL,
;  2600        P 2722  1         OPERATOR_ENTRY('EQ',    EQUAL,          INFIX,   50,  50),
;  2601        P 2723  1         OPERATOR_ENTRY('EQU',   EQUAL,          INFIX,   50,  50),
;  2602        P 2724  1         OPERATOR_ENTRY('NEQ',   NOT_EQUAL,      INFIX,   50,  50),
;  2603        P 2725  1         OPERATOR_ENTRY('NEQU',  NOT_EQUAL,      INFIX,   50,  50),
;  2604        P 2726  1         OPERATOR_ENTRY('GTR',   GTR_THAN,       INFIX,   50,  50),
;  2605        P 2727  1         OPERATOR_ENTRY('GTRU',  GTR_THAN_U,     INFIX,   50,  50),
;  2606        P 2728  1         OPERATOR_ENTRY('GEQ',   GTR_EQUAL,      INFIX,   50,  50),
;  2607        P 2729  1         OPERATOR_ENTRY('GEQU',  GTR_EQUAL_U,    INFIX,   50,  50),
;  2608        P 2730  1         OPERATOR_ENTRY('LSS',   LSS_THAN,       INFIX,   50,  50),
;  2609        P 2731  1         OPERATOR_ENTRY('LSSU',  LSS_THAN_U,     INFIX,   50,  50),
;  2610        P 2732  1         OPERATOR_ENTRY('LEQ',   LSS_EQUAL,      INFIX,   50,  50),
;  2611        P 2733  1         OPERATOR_ENTRY('LEQU',  LSS_EQUAL_U,    INFIX,   50,  50),
;  2612        P 2734  1         OPERATOR_ENTRY('NOT',   BIT_NOT,        PREFIX, 200,  40),
;  2613        P 2735  1         OPERATOR_ENTRY('AND',   BIT_AND,        INFIX,   30,  30),
;  2614        P 2736  1         OPERATOR_ENTRY('OR',    BIT_OR,         INFIX,   20,  20),
;  2615        P 2737  1         OPERATOR_ENTRY('EQV',   BIT_EQV,        INFIX,   10,  10),
;  2616        P 2738  1         OPERATOR_ENTRY('XOR',   BIT_XOR,        INFIX,   10,  10),
;  2617          2739  1         OPERATOR_ENTRY('MOD',   REMAINDER,      INFIX,   70,  70));
;  2618          2740  1
;  2619          2741  1
;  2620          2742  1 ! Define the MACRO Operator Table for operators whose names are composed of
;  2621          2743  1 ! operator characters such as "+", "-", or "*".  This table includes operators
;  2622          2744  1 ! which are part of DEBUG Primary Symbols (such as "\").
;  2623          2745  1 !
;  2624        P 2746  1 OPERATOR_TABLE(MACRO_OPCHAR_OPTBL,
;  2625        P 2747  1         OPERATOR_ENTRY('\',     GLOBAL_SLASH,   PREFIX, 0, 0, PRIMARY),
;  2626        P 2748  1         OPERATOR_ENTRY('\',     BACKSLASH,      INFIX,  0, 0, PRIMARY),
;  2627        P 2749  1
;  2628        P 2750  1         OPERATOR_ENTRY('(',     OPENPAREN,      PREFIX, 200,   5, LEXICAL),
;  2629        P 2751  1         OPERATOR_ENTRY(')',     CLOSEPAREN,     POSTFIX,  6, 200, LEXICAL),
;  2630        P 2752  1         OPERATOR_ENTRY('<',     BITSELECT,      POSTFIX,110, 200, LEXICAL),
;  2631        P 2753  1         OPERATOR_ENTRY('.',     INDIRECT,       PREFIX, 200, 100),
;  2632        P 2754  1         OPERATOR_ENTRY('@',     INDIRECT,       PREFIX, 200, 100),
```

```
;  2633        P 2755  1          OPERATOR_ENTRY('+',     UNARY_PLUS,     PREFIX, 200,  90),
;  2634        P 2756  1          OPERATOR_ENTRY('-',     UNARY_MINUS,    PREFIX, 200,  90),
;  2635        P 2757  1          OPERATOR_ENTRY('@',     LEFT_SHIFT,     INFIX,   80,  80),
;  2636        P 2758  1          OPERATOR_ENTRY('*',     MULTIPLY,       INFIX,   70,  70),
;  2637        P 2759  1          OPERATOR_ENTRY('/',     DIVIDE,         INFIX,   70,  70),
;  2638        P 2760  1          OPERATOR_ENTRY('+',     ADD,            INFIX,   60,  60),
;  2639        P 2761  1          OPERATOR_ENTRY('-',     SUBTRACT,       INFIX,   60,  60)
;  2640          2762  1          );
;  2641          2763  1
;  2642          2764  1
;  2643          2765  1  ! Define the MACRO Terminator Lexical Token Table for subscript expressions.
;  2644          2766  1  ! Since MACRO does not have subscripting, this table is empty.
;  2645          2767  1  !
;  2646          2768  1  TERMINATOR_TABLE(MACRO_SUBSCR_TERM_TBL);
;  2647          2769  1
;  2648          2770  1
;  2649          2771  1  ! Define the MACRO Predefined Identifer Table.
;  2650          2772  1  !
;  2651          2773  1  PRID_TABLE(MACRO_PRID_TABLE);
;  2652          2774  1
;  2653          2775  1
;  2654          2776  1  ! Define the MACRO Built-in Function Table.
;  2655          2777  1  !
;  2656          2778  1  BUILT_IN_FUNCTION_TABLE(MACRO_FUNCTION_TABLE);
;  2657          2779  1
;  2658          2780  1
;  2659          2781  1  ! Define the MACRO Number Scanner State Table.  This table defines the states
;  2660          2782  1  ! of a Finite-State Machine which picks up all valid numeric constants in the
;  2661          2783  1  ! language.
;  2662          2784  1  !
;  2663          2785  1  ! The MACRO number table is the same as the number table for language UNKNOWN.
;  2664          2786  1  !
;  2665          2787  1  BIND
;  2666          2788  1      MACRO_NUMBER_TABLE = UNKNOWN_NUMBER_TABLE;
;  2667          2789  1
;  2668          2790  1
;  2669          2791  1  ! Define the Primary Parser State Table for language MACRO.  Each transition
;  2670          2792  1  ! Entry in the state table has this format:
;  2671          2793  1  !
;  2672          2794  1  !       PRIMARY_TRANSITION(operator-code, action, next-state)
;  2673          2795  1  !
;  2674          2796  1  ! where the first parameter is the operator code which causes the transition
;  2675          2797  1  ! to be taken, the second parameter is the action routine CASE index for the
;  2676          2798  1  ! transition, and the third parameter is the next state in the Finite-State
;  2677          2799  1  ! Machine.
;  2678          2800  1  !
;  2679        P 2801  1  PRIMARY_STATE_TABLE(MACRO_PRIMARY_TABLE,
;  2680        P 2802  1
;  2681        P 2803  1      PRIMARY_STATE(START_STATE,
;  2682        P 2804  1          PRIMARY_TRANSITION(GLOBAL_SLASH, START_GBL, GET_GLOBAL),
;  2683        P 2805  1          PRIMARY_TRANSITION(BACKSLASH, START_SLASH, GOT_BACKSLASH),
;  2684        P 2806  1          PRIMARY_TRANSITION(INVOCNUM, SLASH_INVOCNUM, GOT_BACKSLASH),
;  2685        P 2807  1          PRIMARY_TRANSITION(PRIMARY_TERM, START_TERM, END_STATE)),
;  2686        P 2808  1
;  2687        P 2809  1      PRIMARY_STATE(GET_GLOBAL,
;  2688        P 2810  1          PRIMARY_TRANSITION(PRIMARY_TERM, GBL_TERM, END_STATE))
;  2689        P 2811  1
```

```
;  2690        P 2812  1      PRIMARY_STATE(GOT_BACKSLASH,
;  2691        P 2813  1          PRIMARY_TRANSITION(BACKSLASH, SLASH_SLASH, GOT_BACKSLASH),
;  2692        P 2814  1          PRIMARY_TRANSITION(INVOCNUM, SLASH_INVOCNUM, GOT_BACKSLASH),
;  2693        P 2815  1          PRIMARY_TRANSITION(PRIMARY_TERM, SLASH_TERM, END_STATE)),
;  2694        P 2816  1
;  2695          2817  1      PRIMARY_STATE(END_STATE));
;  2696          2818  1
;  2697          2819  1
;  2698          2820  1  ! Define the table of pointers to the parse tables for MACRO.
;  2699          2821  1  !
;  2700        P 2822  1  LANGUAGE_TABLES(LANGUAGE = MACRO,
;  2701        P 2823  1                  CHARTBL = MACRO_CHARTBL,
;  2702        P 2824  1                  IDENT_OPTBL = MACRO_IDENT_OPTBL,
;  2703        P 2825  1                  OPCHAR_OPTBL = MACRO_OPCHAR_OPTBL,
;  2704        P 2826  1                  NUMBER_TABLE = MACRO_NUMBER_TABLE,
;  2705        P 2827  1                  PRIMARY_TABLE = MACRO_PRIMARY_TABLE,
;  2706        P 2828  1                  SUBSCR_TERMS = MACRO_SUBSCR_TERM_TBL,
;  2707        P 2829  1                  PRIDTBL = MACRO_PRID_TABLE,
;  2708          2830  1                  BIF_TABLE = MACRO_FUNCTION_TABLE);
```

```
;2710          2831  1 !                          P A S C A L   P A R S E   T A B L E S
;2711          2832  1 !
;2712          2833  1 !
;2713          2834  1 !
;2714          2835  1 !       This section includes all the Lexical Scanner and Parser tables needed
;2715          2836  1 !       to scan and parse the PASCAL language.
;2716          2837  1 !
;2717          2838  1 !
;2718          2839  1 !
;2719          2840  1 ! Define the PASCAL Character Table.  What is listed here is actually a list of
;2720          2841  1 ! exceptions to the Character Table for Language UNKNOWN.
;2721          2842  1 !
;2722        P 2843  1 CHAR_EXCEPTION_TABLE(PASCAL_CHARTBL,
;2723        P 2844  1         CHAR_ENTRY('<', OTHER,  OPCHAR, OPCHAR_INFIX, ADDRESS_OP),
;2724        P 2845  1         CHAR_ENTRY('=', OTHER,  OPCHAR, OPCHAR_INFIX, TERMINATOR),
;2725        P 2846  1         CHAR_ENTRY('>', OTHER,  OPCHAR, OPCHAR_INFIX, TERMINATOR),
;2726        P 2847  1         CHAR_ENTRY('^', OTHER,  OPCHAR, SPECIAL_SYMBOL),
;2727        P 2848  1         CHAR_ENTRY('.', DOT,    NUMBER_START, OPCHAR, ADDRESS_OP, SPECIAL_SYMBOL, TERMINATOR),
;2728        P 2849  1         CHAR_ENTRY('[', OTHER,  OPCHAR),
;2729          2850  1         CHAR_ENTRY(']', OTHER,  TERMINATOR));
;2730          2851  1
;2731          2852  1
;2732          2853  1 ! Define the PASCAL Operator Table for operators whose names are identifiers.
;2733          2854  1 !
;2734        P 2855  1 OPERATOR_TABLE(PASCAL_IDENT_OPTBL,
;2735        P 2856  1         OPERATOR_ENTRY('DIV',  INT_DIVIDE,      INFIX,   70,  70),
;2736        P 2857  1         OPERATOR_ENTRY('MOD',  MODULUS,         INFIX,   70,  70),
;2737        P 2858  1         OPERATOR_ENTRY('REM',  REMAINDER,       INFIX,   70,  70),
;2738        P 2859  1         OPERATOR_ENTRY('AND',  AND,             INFIX,   70,  70),
;2739        P 2860  1         OPERATOR_ENTRY('OR',   OR,              INFIX,   60,  60),
;2740        P 2861  1         OPERATOR_ENTRY('NOT',  NOT,             PREFIX, 200,  90),
;2741          2862  1         OPERATOR_ENTRY('IN',   SET_MEMBER,      INFIX,   50,  50));
;2742          2863  1
;2743          2864  1
;2744          2865  1 ! Define the PASCAL Operator Table for operators whose names are composed of
;2745          2866  1 ! operator characters such as "+", "-", or "*".  This table includes operators
;2746          2867  1 ! which are part of DEBUG Primary Symbols (such as "\").
;2747          2868  1 !
;2748        P 2869  1 OPERATOR_TABLE(PASCAL_OPCHAR_OPTBL,
;2749        P 2870  1         OPERATOR_ENTRY('\',    GLOBAL_SLASH,   PREFIX, 0, 0, PRIMARY),
;2750        P 2871  1         OPERATOR_ENTRY('\',    BACKSLASH,      INFIX,  0, 0, PRIMARY),
;2751        P 2872  1         OPERATOR_ENTRY('[',    SUBSCRIPT,      POSTFIX,0, 0, PRIMARY),
;2752        P 2873  1         OPERATOR_ENTRY('.',    DOT,            INFIX,  0, 0, PRIMARY),
;2753        P 2874  1         OPERATOR_ENTRY('^',    PASCAL_DEREF,   POSTFIX,0, 0, PRIMARY),
;2754        P 2875  1         OPERATOR_ENTRY('(',    BIF_OP,         POSTFIX, 0, 0, PRIMARY),
;2755        P 2876  1
;2756        P 2877  1         OPERATOR_ENTRY('(',    OPENPAREN,      PREFIX, 200,   5, LEXICAL),
;2757        P 2878  1         OPERATOR_ENTRY(')',    CLOSEPAREN,     POSTFIX,  6, 200, LEXICAL),
;2758        P 2879  1         OPERATOR_ENTRY('[',    OPENSET,        PREFIX, 200,   5, LEXICAL),
;2759        P 2880  1         OPERATOR_ENTRY('+',    UNARY_PLUS,     PREFIX, 200,  60),
;2760        P 2881  1         OPERATOR_ENTRY('-',    UNARY_MINUS,    PREFIX, 200,  60),
;2761        P 2882  1         OPERATOR_ENTRY('**',   POWER_OF,       INFIX,   80,  80),
;2762        P 2883  1         OPERATOR_ENTRY('*',    MULTIPLY,       INFIX,   70,  70),
;2763        P 2884  1         OPERATOR_ENTRY('/',    DIVIDE,         INFIX,   70,  70),
;2764        P 2885  1         OPERATOR_ENTRY('+',    ADD,            INFIX,   60,  60),
;2765        P 2886  1         OPERATOR_ENTRY('-',    SUBTRACT,       INFIX,   60,  60),
;2766        P 2887  1         OPERATOR_ENTRY('<',    LSS_THAN,       INFIX,   50,  50),
```

```
;  2767        P 2888  1          OPERATOR_ENTRY('<=',    LSS_EQUAL,      INFIX,   50,  50),
;  2768        P 2889  1          OPERATOR_ENTRY('>',     GTR_THAN,       INFIX,   50,  50),
;  2769        P 2890  1          OPERATOR_ENTRY('>=',    GTR_EQUAL,      INFIX,   50,  50),
;  2770        P 2891  1          OPERATOR_ENTRY('=',     EQUAL,          INFIX,   50,  50),
;  2771          2892  1          OPERATOR_ENTRY('<>',    NOT_EQUAL,      INFIX,   50,  50));
;  2772          2893  1
;  2773          2894  1
;  2774          2895  1  ! Define the PASCAL Terminator Lexical Token Table for subscript expressions.
;  2775          2896  1  !
;  2776        P 2897  1  TERMINATOR_TABLE(PASCAL_SUBSCR_TERM_TBL,
;  2777        P 2898  1          TERMINATOR_ENTRY(']', TERM_CLOSE),
;  2778        P 2899  1          TERMINATOR_ENTRY(':', TERM_COLON, MUST_BE_SINGLE),
;  2779          2900  1          TERMINATOR_ENTRY(',', TERM_COMMA));
;  2780          2901  1
;  2781          2902  1
;  2782          2903  1  ! Define the PASCAL Predefined Identifier Table.
;  2783          2904  1  !
;  2784        P 2905  1  PRID_TABLE(PASCAL_PRID_TABLE,
;  2785        P 2906  1          PRID_ENTRY('TRUE', ATOMIC, TF, 1),
;  2786        P 2907  1          PRID_ENTRY('FALSE', ATOMIC, TF, 0),
;  2787          2908  1          PRID_ENTRY('NIL', TPTR, Z, 0));
;  2788          2909  1
;  2789          2910  1
;  2790          2911  1  ! Define the PASCAL Built-in Function Table.
;  2791          2912  1  !
;  2792          2913  1  BUILT_IN_FUNCTION_TABLE(PASCAL_FUNCTION_TABLE);
;  2793          2914  1
;  2794          2915  1
;  2795          2916  1  ! Define the PASCAL Number Scanner State Table.  This table defines the states
;  2796          2917  1  ! of a Finite-State Machine which picks up all valid numeric constants in the
;  2797          2918  1  ! language.
;  2798          2919  1  !
;  2799          2920  1  ! The PASCAL number table is the same as the number table for language UNKNOWN.
;  2800          2921  1  !
;  2801          2922  1  BIND
;  2802          2923  1      PASCAL_NUMBER_TABLE = UNKNOWN_NUMBER_TABLE;
;  2803          2924  1
;  2804          2925  1
;  2805          2926  1  ! Define the Primary Parser State Table for language PASCAL.  Each transition
;  2806          2927  1  ! Entry in the state table has this format:
;  2807          2928  1  !
;  2808          2929  1  !       PRIMARY_TRANSITION(operator-code, action, next-state)
;  2809          2930  1  !
;  2810          2931  1  ! where the first parameter is the operator code which causes the transition
;  2811          2932  1  ! to be taken, the second parameter is the action routine CASE index for the
;  2812          2933  1  ! transition, and the third parameter is the next state in the Finite-State
;  2813          2934  1  ! Machine.
;  2814          2935  1  !
;  2815        P 2936  1  PRIMARY_STATE_TABLE(PASCAL_PRIMARY_TABLE,
;  2816        P 2937  1
;  2817        P 2938  1      PRIMARY_STATE(START_STATE,
;  2818        P 2939  1          PRIMARY_TRANSITION(GLOBAL_SLASH, START_GBL, GET_GLOBAL),
;  2819        P 2940  1          PRIMARY_TRANSITION(BACKSLASH, START_SLASH, GOT_BACKSLASH),
;  2820        P 2941  1          PRIMARY_TRANSITION(INVOCNUM, SLASH_INVOCNUM, GOT_BACKSLASH),
;  2821        P 2942  1          PRIMARY_TRANSITION(DOT, START_DOT, GOT_DOT),
;  2822        P 2943  1          PRIMARY_TRANSITION(SUBSCRIPT, START_SUBSCR, GOT_SUBSCRIPT),
;  2823        P 2944  1          PRIMARY_TRANSITION(PASCAL_DEREF, START_DEREF, GOT_DEREF),
```

```
;  2824        P 2945  1          PRIMARY_TRANSITION(BIF_OP, START_BIF_CALL, END_STATE),
;  2825        P 2946  1          PRIMARY_TRANSITION(PRIMARY_TERM, START_TERM, END_STATE)),
;  2826        P 2947  1
;  2827        P 2948  1      PRIMARY_STATE(GET_GLOBAL,
;  2828        P 2949  1          PRIMARY_TRANSITION(PRIMARY_TERM, GBL_TERM, END_STATE)),
;  2829        P 2950  1
;  2830        P 2951  1      PRIMARY_STATE(GOT_BACKSLASH,
;  2831        P 2952  1          PRIMARY_TRANSITION(BACKSLASH, SLASH_SLASH, GOT_BACKSLASH),
;  2832        P 2953  1          PRIMARY_TRANSITION(INVOCNUM, SLASH_INVOCNUM, GOT_BACKSLASH),
;  2833        P 2954  1          PRIMARY_TRANSITION(DOT, SLASH_DOT, GOT_DOT),
;  2834        P 2955  1          PRIMARY_TRANSITION(SUBSCRIPT, SLASH_SUBSCR, GOT_SUBSCRIPT),
;  2835        P 2956  1          PRIMARY_TRANSITION(PASCAL_DEREF, SLASH_DEREF, GOT_DEREF),
;  2836        P 2957  1          PRIMARY_TRANSITION(PRIMARY_TERM, SLASH_TERM, END_STATE)),
;  2837        P 2958  1
;  2838        P 2959  1      PRIMARY_STATE(GOT_DOT,
;  2839        P 2960  1          PRIMARY_TRANSITION(DOT, DOT_DOT, GOT_DOT),
;  2840        P 2961  1          PRIMARY_TRANSITION(SUBSCRIPT, DOT_SUBSCR, GOT_SUBSCRIPT),
;  2841        P 2962  1          PRIMARY_TRANSITION(PASCAL_DEREF, DOT_DEREF, GOT_DEREF),
;  2842        P 2963  1          PRIMARY_TRANSITION(PRIMARY_TERM, DOT_TERM, END_STATE)),
;  2843        P 2964  1
;  2844        P 2965  1      PRIMARY_STATE(GOT_SUBSCRIPT,
;  2845        P 2966  1          PRIMARY_TRANSITION(DOT, SUBSCR_DOT, GOT_DOT),
;  2846        P 2967  1          PRIMARY_TRANSITION(SUBSCRIPT, SUBSCR_SUBSCR, GOT_SUBSCRIPT),
;  2847        P 2968  1          PRIMARY_TRANSITION(PASCAL_DEREF, SUBSCR_DEREF, GOT_DEREF),
;  2848        P 2969  1          PRIMARY_TRANSITION(PRIMARY_TERM, SUBSCR_TERM, END_STATE)),
;  2849        P 2970  1
;  2850        P 2971  1      PRIMARY_STATE(GOT_DEREF,
;  2851        P 2972  1          PRIMARY_TRANSITION(DOT, DEREF_DOT, GOT_DOT),
;  2852        P 2973  1          PRIMARY_TRANSITION(SUBSCRIPT, DEREF_SUBSCR, GOT_SUBSCRIPT),
;  2853        P 2974  1          PRIMARY_TRANSITION(PASCAL_DEREF, DEREF_DEREF, GOT_DEREF),
;  2854        P 2975  1          PRIMARY_TRANSITION(PRIMARY_TERM, DEREF_TERM, END_STATE)),
;  2855        P 2976  1
;  2856          2977  1      PRIMARY_STATE(END_STATE));
;  2857          2978  1
;  2858          2979  1
;  2859          2980  1  ! Define the table of pointers to the parse tables for PASCAL.
;  2860          2981  1  !
;  2861        P 2982  1  LANGUAGE_TABLES(LANGUAGE = PASCAL,
;  2862        P 2983  1                  CHARTBL = PASCAL_CHARTBL,
;  2863        P 2984  1                  IDENT_OPTBL = PASCAL_IDENT_OPTBL,
;  2864        P 2985  1                  OPCHAR_OPTBL = PASCAL_OPCHAR_OPTBL,
;  2865        P 2986  1                  NUMBER_TABLE = PASCAL_NUMBER_TABLE,
;  2866        P 2987  1                  PRIMARY_TABLE = PASCAL_PRIMARY_TABLE,
;  2867        P 2988  1                  SUBSCR_TERMS = PASCAL_SUBSCR_TERM_TBL,
;  2868        P 2989  1                  PRIDTBL = PASCAL_PRID_TABLE,
;  2869        P 2990  1                  BIF_TABLE = PASCAL_FUNCTION_TABLE,
;  2870          2991  1                  MULTIPLE_SUBSCR = TRUE);
```

```
;  2872          2992  1 !                          P L / I   P A R S E   T A B L E S
;  2873          2993  1 !
;  2874          2994  1 !
;  2875          2995  1 !
;  2876          2996  1 !       This section includes all the Lexical Scanner and Parser tables needed
;  2877          2997  1 !       to scan and parse the PL/I language.
;  2878          2998  1 !
;  2879          2999  1 !
;  2880          3000  1 !
;  2881          3001  1 ! Define the PL/I Character Table.  What is listed here is actually a list of
;  2882          3002  1 ! exceptions to the Character Table for Language UNKNOWN.
;  2883          3003  1 !
;  2884        P 3004  1 CHAR_EXCEPTION_TABLE(PLI_CHARTBL,
;  2885        P 3005  1         CHAR_ENTRY('$', OTHER,  IDENT_ANYWHERE),
;  2886        P 3006  1         CHAR_ENTRY('_', OTHER,  IDENT_ANYWHERE),
;  2887        P 3007  1         CHAR_ENTRY('%', OTHER,  IDENT_MIDDLE),
;  2888        P 3008  1         CHAR_ENTRY('&', OTHER,  OPCHAR),
;  2889        P 3009  1         CHAR_ENTRY('|', OTHER,  OPCHAR, OPCHAR_INFIX),
;  2890        P 3010  1         CHAR_ENTRY('!', OTHER,  OPCHAR, OPCHAR_INFIX),
;  2891        P 3011  1         CHAR_ENTRY('=', OTHER,  OPCHAR, OPCHAR_INFIX, TERMINATOR),
;  2892        P 3012  1         CHAR_ENTRY('^', OTHER,  OPCHAR, OPCHAR_INFIX, SPECIAL_SYMBOL),
;  2893        P 3013  1         CHAR_ENTRY('-', MINUS,  OPCHAR, ADDRESS_OP, SPECIAL_CASE),
;  2894        P 3014  1         CHAR_ENTRY('<', OTHER,  OPCHAR, OPCHAR_INFIX, ADDRESS_OP),
;  2895          3015  1         CHAR_ENTRY('>', OTHER,  OPCHAR, OPCHAR_INFIX, TERMINATOR));
;  2896          3016  1
;  2897          3017  1
;  2898          3018  1 ! Define the PL/I Operator Table for operators whose names are identifiers.
;  2899          3019  1 !
;  2900          3020  1 OPERATOR_TABLE(PLI_IDENT_OPTBL);
;  2901          3021  1
;  2902          3022  1
;  2903          3023  1 ! Define the PL/I Operator Table for operators whose names are composed of
;  2904          3024  1 ! operator characters such as "+", "-", or "*".  This table includes operators
;  2905          3025  1 ! which are part of DEBUG Primary Symbols (such as "\").
;  2906          3026  1 !
;  2907        P 3027  1 OPERATOR_TABLE(PLI_OPCHAR_OPTBL,
;  2908        P 3028  1         OPERATOR_ENTRY('\',     GLOBAL_SLASH,   PREFIX, 0, 0, PRIMARY),
;  2909        P 3029  1         OPERATOR_ENTRY('\',     BACKSLASH,      INFIX,  0, 0, PRIMARY),
;  2910        P 3030  1         OPERATOR_ENTRY('(',     SUBSCRIPT,      POSTFIX,0, 0, PRIMARY),
;  2911        P 3031  1         OPERATOR_ENTRY('.',     DOT,            INFIX,  0, 0, PRIMARY),
;  2912        P 3032  1
;  2913        P 3033  1         OPERATOR_ENTRY('(',     OPENPAREN,      PREFIX, 200,   5, LEXICAL),
;  2914        P 3034  1         OPERATOR_ENTRY(')',     CLOSEPAREN,     POSTFIX,  6, 200, LEXICAL),
;  2915        P 3035  1         OPERATOR_ENTRY('+',     UNARY_PLUS,     PREFIX, 200, 70),
;  2916        P 3036  1         OPERATOR_ENTRY('-',     UNARY_MINUS,    PREFIX, 200, 70),
;  2917        P 3037  1         OPERATOR_ENTRY('^',     BIT_NOT,        PREFIX, 200, 30),
;  2918        P 3038  1         OPERATOR_ENTRY('**',    POWER_OF,       INFIX,  92,  90),
;  2919        P 3039  1         OPERATOR_ENTRY('*',     MULTIPLY,       INFIX,  80,  80),
;  2920        P 3040  1         OPERATOR_ENTRY('/',     DIVIDE,         INFIX,  80,  80),
;  2921        P 3041  1         OPERATOR_ENTRY('+',     ADD,            INFIX,  60,  60),
;  2922        P 3042  1         OPERATOR_ENTRY('-',     SUBTRACT,       INFIX,  60,  60),
;  2923        P 3043  1         OPERATOR_ENTRY('||',    CONCATENATE,    INFIX,  55,  55),
;  2924        P 3044  1         OPERATOR_ENTRY('>',     GTR_THAN,       INFIX,  50,  50),
;  2925        P 3045  1         OPERATOR_ENTRY('<',     LSS_THAN,       INFIX,  50,  50),
;  2926        P 3046  1         OPERATOR_ENTRY('^>',    LSS_EQUAL,      INFIX,  50,  50),
;  2927        P 3047  1         OPERATOR_ENTRY('^<',    GTR_EQUAL,      INFIX,  50,  50),
;  2928        P 3048  1         OPERATOR_ENTRY('=',     EQUAL,          INFIX,  50,  50),
```

```
;2929        P 3049  1          OPERATOR_ENTRY('^=',    NOT_EQUAL,      INFIX,  50,  50),
;2930        P 3050  1          OPERATOR_ENTRY('<=',    LSS_EQUAL,      INFIX,  50,  50),
;2931        P 3051  1          OPERATOR_ENTRY('>=',    GTR_EQUAL,      INFIX,  50,  50),
;2932        P 3052  1          OPERATOR_ENTRY('&',     BIT_AND,        INFIX,  45,  45),
;2933          3053  1          OPERATOR_ENTRY('|',     BIT_OR,         INFIX,  40,  40));
;2934          3054  1
;2935          3055  1
;2936          3056  1  ! Define the Lexical Token Entry for the PL/I dereference operator "->".
;2937          3057  1  ! This token is lexically scanned separately.
;2938          3058  1  !
;2939          3059  1  BIND
;2940          3060  1      PLI_ARROW_TOKEN =
;2941          3061  1          OPERATOR_ENTRY('->',    PLI_DEREF,      INFIX,  0, 0, PRIMARY);
;2942          3062  1
;2943          3063  1
;2944          3064  1  ! Define the PL/I Terminator Lexical Token Table for subscript expressions.
;2945          3065  1  !
;2946        P 3066  1  TERMINATOR_TABLE(PLI_SUBSCR_TERM_TBL,
;2947        P 3067  1          TERMINATOR_ENTRY(')', TERM_CLOSE),
;2948        P 3068  1          TERMINATOR_ENTRY(':', TERM_COLON, MUST_BE_SINGLE),
;2949          3069  1          TERMINATOR_ENTRY(',', TERM_COMMA));
;2950          3070  1
;2951          3071  1
;2952          3072  1  ! Define the PL/I Predefined Identifier Table.
;2953          3073  1  !
;2954          3074  1  PRID_TABLE(PLI_PRID_TABLE);
;2955          3075  1
;2956          3076  1
;2957          3077  1  ! Define the PLI Built-in Function Table.
;2958          3078  1  !
;2959          3079  1  BUILT_IN_FUNCTION_TABLE(PLI_FUNCTION_TABLE);
;2960          3080  1
;2961          3081  1
;2962          3082  1  ! Define the PLI Number Scanner State Table.  This table defines the states
;2963          3083  1  ! of a Finite-State Machine which picks up all valid numeric constants in the
;2964          3084  1  ! language.
;2965          3085  1  !
;2966          3086  1  ! Define the language PLI Number Scanner State Table. This is a finite-state
;2967          3087  1  ! machine in which each transition is of the form:
;2968          3088  1  !
;2969          3089  1  !       NUMBER_TRANSITION(character-class, action-index, next-state)
;2970          3090  1  !
;2971          3091  1  ! where the character-class and action-index names are automatically prefixed
;2972          3092  1  ! by 'NUMST$K_CLASS_' or 'NUMST$K_ACT_' by the NUMBER_TRANSITION macro.
;2973          3093  1  !
;2974        P 3094  1  NUMBER_STATE_TABLE(PLI_NUMBER_TABLE,
;2975        P 3095  1
;2976        P 3096  1      NUMBER_STATE(START_STATE,
;2977        P 3097  1          NUMBER_TRANSITION(DIGIT, GO_PAST_PACK, ACCUM_INT),
;2978        P 3098  1          NUMBER_TRANSITION(DOT, DO_NOTHING, LEADING_DOT),
;2979        P 3099  1          NUMBER_TRANSITION(OTHER, NOT_NUMBER, END_STATE)),
;2980        P 3100  1
;2981        P 3101  1      NUMBER_STATE(LEADING_DOT,
;2982        P 3102  1          NUMBER_TRANSITION(DIGIT, GO_PAST_PACK_FRAC, ACCUM_FRAC),
;2983        P 3103  1          NUMBER_TRANSITION(OTHER, NOT_NUMBER, END_STATE)),
;2984        P 3104  1
;2985        P 3105  1      NUMBER_STATE(ACCUM_INT,
```

```
;  2986        P 3106  1          NUMBER_TRANSITION(DIGIT, GO_PAST_PACK, ACCUM_INT),
;  2987        P 3107  1          NUMBER_TRANSITION(DOT, DO_NOTHING, ACCUM_FRAC),
;  2988        P 3108  1          NUMBER_TRANSITION(HEXDIGIT, DO_NOTHING, ACCUM_HEX),
;  2989        P 3109  1          NUMBER_TRANSITION(B, DO_NOTHING, ACCUM_HEX),
;  2990        P 3110  1          NUMBER_TRANSITION(D, DO_NOTHING, ACCUM_HEX),
;  2991        P 3111  1          NUMBER_TRANSITION(E, DO_NOTHING, ACCUM_HEX),
;  2992        P 3112  1          NUMBER_TRANSITION(OTHER, GOT_PACK_NUMBER, END_STATE)),
;  2993        P 3113  1
;  2994        P 3114  1      NUMBER_STATE(ACCUM_HEX,
;  2995        P 3115  1          NUMBER_TRANSITION(DIGIT, DO_NOTHING, ACCUM_HEX),
;  2996        P 3116  1          NUMBER_TRANSITION(HEXDIGIT, DO_NOTHING, ACCUM_HEX),
;  2997        P 3117  1          NUMBER_TRANSITION(B, DO_NOTHING, ACCUM_HEX),
;  2998        P 3118  1          NUMBER_TRANSITION(D, DO_NOTHING, ACCUM_HEX),
;  2999        P 3119  1          NUMBER_TRANSITION(E, DO_NOTHING, ACCUM_HEX),
;  3000        P 3120  1          NUMBER_TRANSITION(OTHER, GOT_NUMBER, END_STATE)),
;  3001        P 3121  1
;  3002        P 3122  1      NUMBER_STATE(ACCUM_FRAC,
;  3003        P 3123  1          NUMBER_TRANSITION(DIGIT, GO_PAST_PACK_FRAC, ACCUM_FRAC),
;  3004        P 3124  1          NUMBER_TRANSITION(DOT, BACKUP_PTRS, END_STATE),
;  3005        P 3125  1          NUMBER_TRANSITION(E, MARK_E_EXP, GET_EXPONENT),
;  3006        P 3126  1          NUMBER_TRANSITION(D, MARK_D_EXP, GET_EXPONENT),
;  3007        P 3127  1          NUMBER_TRANSITION(G, MARK_G_EXP, GET_EXPONENT),
;  3008        P 3128  1          NUMBER_TRANSITION(Q, MARK_Q_EXP, GET_EXPONENT),
;  3009        P 3129  1          NUMBER_TRANSITION(OTHER, GOT_PACK_NUMBER, END_STATE)),
;  3010        P 3130  1
;  3011        P 3131  1      NUMBER_STATE(GET_EXPONENT,
;  3012        P 3132  1          NUMBER_TRANSITION(DIGIT, DO_NOTHING, ACCUM_EXP),
;  3013        P 3133  1          NUMBER_TRANSITION(PLUS, DO_NOTHING, GET_EXP_SIGN),
;  3014        P 3134  1          NUMBER_TRANSITION(MINUS, DO_NOTHING, GET_EXP_SIGN),
;  3015        P 3135  1          NUMBER_TRANSITION(OTHER, BACKUP_PTRS, END_STATE)),
;  3016        P 3136  1
;  3017        P 3137  1      NUMBER_STATE(GET_EXP_SIGN,
;  3018        P 3138  1          NUMBER_TRANSITION(DIGIT, DO_NOTHING, ACCUM_EXP),
;  3019        P 3139  1          NUMBER_TRANSITION(OTHER, GOT_NUMBER, END_STATE)),
;  3020        P 3140  1
;  3021        P 3141  1      NUMBER_STATE(ACCUM_EXP,
;  3022        P 3142  1          NUMBER_TRANSITION(DIGIT, DO_NOTHING, ACCUM_EXP),
;  3023        P 3143  1          NUMBER_TRANSITION(OTHER, GOT_NUMBER, END_STATE)),
;  3024        P 3144  1
;  3025        P 3145  1      NUMBER_STATE(END_STATE,
;  3026          3146  1          NUMBER_TRANSITION(OTHER, GIVE_ERROR, END_STATE)));
;  3027          3147  1
;  3028          3148  1
;  3029          3149  1  ! Define the Primary Parser State Table for language PL/I.  Each transition
;  3030          3150  1  ! Entry in the state table has this format:
;  3031          3151  1  !
;  3032          3152  1  !       PRIMARY_TRANSITION(operator-code, action, next-state)
;  3033          3153  1  !
;  3034          3154  1  ! where the first parameter is the operator code which causes the transition
;  3035          3155  1  ! to be taken, the second parameter is the action routine CASE index for the
;  3036          3156  1  ! transition, and the third parameter is the next state in the Finite-State
;  3037          3157  1  ! Machine.
;  3038          3158  1  !
;  3039        P 3159  1  PRIMARY_STATE_TABLE(PLI_PRIMARY_TABLE,
;  3040        P 3160  1
;  3041        P 3161  1      PRIMARY_STATE(START_STATE,
;  3042        P 3162  1          PRIMARY_TRANSITION(GLOBAL_SLASH, START_GBL, GET_GLOBAL),
```

```
;  3043        P 3163  1          PRIMARY_TRANSITION(BACKSLASH, START_SLASH, GOT_BACKSLASH),
;  3044        P 3164  1          PRIMARY_TRANSITION(INVOCNUM, SLASH_INVOCNUM, GOT_BACKSLASH),
;  3045        P 3165  1          PRIMARY_TRANSITION(DOT, START_DOT_PLI, GOT_DOT),
;  3046        P 3166  1          PRIMARY_TRANSITION(SUBSCRIPT, START_SUBSCR_PLI, GOT_SUBSCRIPT),
;  3047        P 3167  1          PRIMARY_TRANSITION(PLI_DEREF, START_DEREF_PLI, START_STATE),
;  3048        P 3168  1          PRIMARY_TRANSITION(PRIMARY_TERM, START_TERM, END_STATE)),
;  3049        P 3169  1
;  3050        P 3170  1      PRIMARY_STATE(GET_GLOBAL,
;  3051        P 3171  1          PRIMARY_TRANSITION(PRIMARY_TERM, GBL_TERM, END_STATE)),
;  3052        P 3172  1
;  3053        P 3173  1      PRIMARY_STATE(GOT_BACKSLASH,
;  3054        P 3174  1          PRIMARY_TRANSITION(BACKSLASH, SLASH_SLASH, GOT_BACKSLASH),
;  3055        P 3175  1          PRIMARY_TRANSITION(INVOCNUM, SLASH_INVOCNUM, GOT_BACKSLASH),
;  3056        P 3176  1          PRIMARY_TRANSITION(DOT, SLASH_DOT_PLI, GOT_DOT),
;  3057        P 3177  1          PRIMARY_TRANSITION(SUBSCRIPT, SLASH_SUBSCR_PLI, GOT_SUBSCRIPT),
;  3058        P 3178  1          PRIMARY_TRANSITION(PLI_DEREF, SLASH_DEREF_PLI, START_STATE),
;  3059        P 3179  1          PRIMARY_TRANSITION(PRIMARY_TERM, SLASH_TERM, END_STATE)),
;  3060        P 3180  1
;  3061        P 3181  1      PRIMARY_STATE(GOT_DOT,
;  3062        P 3182  1          PRIMARY_TRANSITION(DOT, DOT_DOT_PLI, GOT_DOT),
;  3063        P 3183  1          PRIMARY_TRANSITION(SUBSCRIPT, DOT_SUBSCR_PLI, GOT_SUBSCRIPT),
;  3064        P 3184  1          PRIMARY_TRANSITION(PLI_DEREF, DOT_DEREF_PLI, START_STATE),
;  3065        P 3185  1          PRIMARY_TRANSITION(PRIMARY_TERM, DOT_TERM_PLI, END_STATE)),
;  3066        P 3186  1
;  3067        P 3187  1      PRIMARY_STATE(GOT_SUBSCRIPT,
;  3068        P 3188  1          PRIMARY_TRANSITION(DOT, SUBSCR_DOT_PLI, GOT_DOT),
;  3069        P 3189  1          PRIMARY_TRANSITION(SUBSCRIPT, SUBSCR_SUBSCR_PLI, GOT_SUBSCRIPT),
;  3070        P 3190  1          PRIMARY_TRANSITION(PLI_DEREF, SUBSCR_DEREF_PLI, START_STATE),
;  3071        P 3191  1          PRIMARY_TRANSITION(PRIMARY_TERM, SUBSCR_TERM_PLI, END_STATE)),
;  3072        P 3192  1
;  3073          3193  1      PRIMARY_STATE(END_STATE));
;  3074          3194  1
;  3075          3195  1
;  3076          3196  1  ! Define the table of pointers to the parse tables for PL/I.
;  3077          3197  1  !
;  3078        P 3198  1  LANGUAGE_TABLES(LANGUAGE = PLI,
;  3079        P 3199  1                  CHARTBL = PLI_CHARTBL,
;  3080        P 3200  1                  IDENT_OPTBL = PLI_IDENT_OPTBL,
;  3081        P 3201  1                  OPCHAR_OPTBL = PLI_OPCHAR_OPTBL,
;  3082        P 3202  1                  NUMBER_TABLE = PLI_NUMBER_TABLE,
;  3083        P 3203  1                  PRIMARY_TABLE = PLI_PRIMARY_TABLE,
;  3084        P 3204  1                  SUBSCR_TERMS = PLI_SUBSCR_TERM_TBL,
;  3085        P 3205  1                  PRIDTBL = PLI_PRID_TABLE,
;  3086        P 3206  1                  BIF_TABLE = PLI_FUNCTION_TABLE,
;  3087        P 3207  1                  MULTIPLE_SUBSCR = FALSE,
;  3088          3208  1                  COMPONENTS_IN_PATHNAME = TRUE);
```

```
;  3090          3209  1 !                          R P G   P A R S E   T A B L E S
;  3091          3210  1 !
;  3092          3211  1 !
;  3093          3212  1 !
;  3094          3213  1 !       This section includes all the Lexical Scanner and Parser tables needed
;  3095          3214  1 !       to scan and parse the RPG language.
;  3096          3215  1 !
;  3097          3216  1 !
;  3098          3217  1 ! Define the RPG Character Table.  What is listed here is actually a list of
;  3099          3218  1 ! exceptions to the Character Table for Language UNKNOWN.
;  3100          3219  1 !
;  3101        P 3220  1 CHAR_EXCEPTION_TABLE(RPG_CHARTBL,
;  3102        P 3221  1         CHAR_ENTRY('*', OTHER,  NOTHING),
;  3103        P 3222  1         CHAR_ENTRY('<', OTHER,  OPCHAR, OPCHAR_INFIX, ADDRESS_OP, TERMINATOR),
;  3104        P 3223  1         CHAR_ENTRY('>', OTHER,  OPCHAR, OPCHAR_INFIX, TERMINATOR),
;  3105        P 3224  1         CHAR_ENTRY('=', OTHER,  OPCHAR, OPCHAR_INFIX, TERMINATOR),
;  3106        P 3225  1         CHAR_ENTRY('#', OTHER,  IDENT_ANYWHERE),
;  3107        P 3226  1         CHAR_ENTRY('_', OTHER,  IDENT_ANYWHERE),
;  3108          3227  1         CHAR_ENTRY('$', OTHER,  IDENT_ANYWHERE));
;  3109          3228  1
;  3110          3229  1
;  3111          3230  1 ! Define the RPG Operator Table for operators whose names are identifiers.
;  3112          3231  1 !
;  3113        P 3232  1 OPERATOR_TABLE(RPG_IDENT_OPTBL,
;  3114        P 3233  1         OPERATOR_ENTRY('NOT',   NOT,          PREFIX, 200, 11),
;  3115        P 3234  1         OPERATOR_ENTRY('AND',   AND,          INFIX,   10, 10),
;  3116        P 3235  1         OPERATOR_ENTRY('OR',    OR,           INFIX,   10, 10),
;  3117          3236  1         OPERATOR_ENTRY('NOT',   INFIX_NOT,    INFIX,   15, 15, LEXICAL));
;  3118          3237  1
;  3119          3238  1 BIND
;  3120          3239  1         RPG_NOT_EQL_TOKEN =
;  3121          3240  1             OPERATOR_ENTRY('NOT =', NOT_EQUAL,    INFIX,   15, 15),
;  3122          3241  1         RPG_NOT_GTR_TOKEN =
;  3123          3242  1             OPERATOR_ENTRY('NOT >', LSS_EQUAL,    INFIX,   15, 15),
;  3124          3243  1         RPG_NOT_LSS_TOKEN =
;  3125          3244  1             OPERATOR_ENTRY('NOT <', GTR_EQUAL,    INFIX,   15, 15);
;  3126          3245  1
;  3127          3246  1
;  3128          3247  1 ! Define the RPG Operator Table for operators whose names are composed of
;  3129          3248  1 ! operator characters such as "+", "-", or "*".  This table includes operators
;  3130          3249  1 ! which are part of DEBUG Primary Symbols (such as "\").
;  3131          3250  1 !
;  3132        P 3251  1 OPERATOR_TABLE(RPG_OPCHAR_OPTBL,
;  3133        P 3252  1         OPERATOR_ENTRY('\',     GLOBAL_SLASH,   PREFIX, 0, 0, PRIMARY),
;  3134        P 3253  1         OPERATOR_ENTRY('\',     BACKSLASH,      INFIX,  0, 0, PRIMARY),
;  3135        P 3254  1         OPERATOR_ENTRY('(',     SUBSCRIPT,      POSTFIX,0, 0, PRIMARY),
;  3136        P 3255  1         OPERATOR_ENTRY('>',     PREFIX_GTR,     PREFIX, 200,  15, LEXICAL),
;  3137        P 3256  1         OPERATOR_ENTRY('<',     PREFIX_LSS,     PREFIX, 200,  15, LEXICAL),
;  3138        P 3257  1         OPERATOR_ENTRY('=',     PREFIX_EQL,     PREFIX, 200,  15, LEXICAL),
;  3139        P 3258  1         OPERATOR_ENTRY('(',     OPENPAREN,      PREFIX, 200,   5, LEXICAL),
;  3140        P 3259  1         OPERATOR_ENTRY(')',     CLOSEPAREN,     POSTFIX,  6, 200, LEXICAL),
;  3141        P 3260  1         OPERATOR_ENTRY('/',     DIVIDE,         INFIX,   30,  30),
;  3142        P 3261  1         OPERATOR_ENTRY('+',     UNARY_PLUS,     PREFIX, 200,  25),
;  3143        P 3262  1         OPERATOR_ENTRY('-',     UNARY_MINUS,    PREFIX, 200,  25),
;  3144        P 3263  1         OPERATOR_ENTRY('+',     ADD,            INFIX,   20,  20),
;  3145        P 3264  1         OPERATOR_ENTRY('-',     SUBTRACT,       INFIX,   20,  20),
;  3146        P 3265  1         OPERATOR_ENTRY('>',     GTR_THAN,       INFIX,   15,  15),
```

```
;  3147        P 3266  1          OPERATOR_ENTRY('<',     LSS_THAN,        INFIX,   15,  15),
;  3148          3267  1          OPERATOR_ENTRY('=',     EQUAL,           INFIX,   15,  15));
;  3149          3268  1
;  3150          3269  1 BIND
;  3151          3270  1     RPG_MULTIPLY_TOKEN =
;  3152          3271  1         OPERATOR_ENTRY('*',     MULTIPLY,        INFIX,   30,  30);
;  3153          3272  1
;  3154          3273  1
;  3155          3274  1 ! Define the RPG Terminator Lexical Token Table for subscript expressions.
;  3156          3275  1 !
;  3157        P 3276  1 TERMINATOR_TABLE(RPG_SUBSCR_TERM_TBL,
;  3158        P 3277  1          TERMINATOR_ENTRY(')', TERM_CLOSE),
;  3159        P 3278  1          TERMINATOR_ENTRY(':', TERM_COLON, MUST_BE_SINGLE),
;  3160          3279  1          TERMINATOR_ENTRY(',', TERM_COMMA));
;  3161          3280  1
;  3162          3281  1
;  3163          3282  1 ! Define the RPG Predefined Identifier Table.
;  3164          3283  1 !
;  3165          3284  1 PRID_TABLE(RPG_PRID_TABLE);
;  3166          3285  1
;  3167          3286  1
;  3168          3287  1 ! Define the RPG Built-in Function Table.
;  3169          3288  1 !
;  3170          3289  1 BUILT_IN_FUNCTION_TABLE(RPG_FUNCTION_TABLE);
;  3171          3290  1
;  3172          3291  1
;  3173          3292  1 ! Define the RPG Number Scanner State Table.  This table defines the states
;  3174          3293  1 ! of a Finite-State Machine which picks up all valid numeric constants in the
;  3175          3294  1 ! language.  Each Transition Entry is of the form:
;  3176          3295  1 !
;  3177          3296  1 !       NUMBER_TRANSITION(character-class, action-index, next-state)
;  3178          3297  1 !
;  3179          3298  1 ! where the character-class and action-index names are automatically prefixed
;  3180          3299  1 ! by 'NUMST$K_CLASS_' or 'NUMST$K_ACT_' by the NUMBER_TRANSITION macro.
;  3181          3300  1 !
;  3182          3301  1 BIND
;  3183          3302  1     RPG_NUMBER_TABLE = PLI_NUMBER_TABLE;
;  3184          3303  1
;  3185          3304  1
;  3186          3305  1 ! Define the Primary Parser State Table for language RPG.  Each transition
;  3187          3306  1 ! Entry in the state table has this format:
;  3188          3307  1 !
;  3189          3308  1 !       PRIMARY_TRANSITION(operator-code, action, next-state)
;  3190          3309  1 !
;  3191          3310  1 ! where the first parameter is the operator code which causes the transition
;  3192          3311  1 ! to be taken, the second parameter is the action routine CASE index for the
;  3193          3312  1 ! transition, and the third parameter is the next state in the Finite-State
;  3194          3313  1 ! Machine.
;  3195          3314  1 !
;  3196        P 3315  1 PRIMARY_STATE_TABLE(RPG_PRIMARY_TABLE,
;  3197        P 3316  1
;  3198        P 3317  1     PRIMARY_STATE(START_STATE,
;  3199        P 3318  1         PRIMARY_TRANSITION(GLOBAL_SLASH, START_GBL, GET_GLOBAL),
;  3200        P 3319  1         PRIMARY_TRANSITION(BACKSLASH, START_SLASH, GOT_BACKSLASH),
;  3201        P 3320  1         PRIMARY_TRANSITION(INVOCNUM, SLASH_INVOCNUM, GOT_BACKSLASH),
;  3202        P 3321  1         PRIMARY_TRANSITION(SUBSCRIPT, START_SUBSCR, GOT_SUBSCRIPT),
;  3203        P 3322  1         PRIMARY_TRANSITION(PRIMARY_TERM, START_TERM, END_STATE)),
```

```
;  3204        P 3323  1
;  3205        P 3324  1     PRIMARY_STATE(GET_GLOBAL,
;  3206        P 3325  1         PRIMARY_TRANSITION(PRIMARY_TERM, GBL_TERM, END_STATE)),
;  3207        P 3326  1
;  3208        P 3327  1     PRIMARY_STATE(GOT_BACKSLASH,
;  3209        P 3328  1         PRIMARY_TRANSITION(BACKSLASH, SLASH_SLASH, GOT_BACKSLASH),
;  3210        P 3329  1         PRIMARY_TRANSITION(INVOCNUM, SLASH_INVOCNUM, GOT_BACKSLASH),
;  3211        P 3330  1         PRIMARY_TRANSITION(SUBSCRIPT, SLASH_SUBSCR, GOT_SUBSCRIPT),
;  3212        P 3331  1         PRIMARY_TRANSITION(PRIMARY_TERM, SLASH_TERM, END_STATE)),
;  3213        P 3332  1
;  3214        P 3333  1     PRIMARY_STATE(GOT_SUBSCRIPT,
;  3215        P 3334  1         PRIMARY_TRANSITION(SUBSCRIPT, SUBSCR_SUBSCR, GOT_SUBSCRIPT),
;  3216        P 3335  1         PRIMARY_TRANSITION(PRIMARY_TERM, SUBSCR_TERM, END_STATE)),
;  3217        P 3336  1
;  3218          3337  1     PRIMARY_STATE(END_STATE));
;  3219          3338  1
;  3220          3339  1
;  3221          3340  1 ! Define the table of pointers to the parse tables for RPG.
;  3222          3341  1 !
;  3223        P 3342  1 LANGUAGE_TABLES(LANGUAGE = RPG,
;  3224        P 3343  1                 CHARTBL = RPG_CHARTBL,
;  3225        P 3344  1                 IDENT_OPTBL = RPG_IDENT_OPTBL,
;  3226        P 3345  1                 OPCHAR_OPTBL = RPG_OPCHAR_OPTBL,
;  3227        P 3346  1                 NUMBER_TABLE = RPG_NUMBER_TABLE,
;  3228        P 3347  1                 PRIMARY_TABLE = RPG_PRIMARY_TABLE,
;  3229        P 3348  1                 SUBSCR_TERMS = RPG_SUBSCR_TERM_TBL,
;  3230        P 3349  1                 PRIDTBL = RPG_PRID_TABLE,
;  3231          3350  1                 BIF_TABLE = PASCAL_FUNCTION_TABLE);
```

```
;  3233          3351  1 !       T A B L E   O F   P O I N T E R S   T O   L A N G U A G E   T A B L E S
;  3234          3352  1 !
;  3235          3353  1 !
;  3236          3354  1 !
;  3237          3355  1 !       This section contains the table of pointers to the language-specific
;  3238          3356  1 !       tables of parser table pointers.  In other words, this table in indexed
;  3239          3357  1 !       by language code to yield a pointer (relative to TABLEBASE) to the
;  3240          3358  1 !       table of table pointers built by the LANGUAGE_TABLES macro for the
;  3241          3359  1 !       individual language above.
;  3242          3360  1 !
;  3243          3361  1 !
;  3244          3362  1 ! Define the table of pointers to language-specific tables.
;  3245          3363  1 !
;  3246          3364  1 OWN
;  3247          3365  1     LANGUAGE_TABLE_PTRS:
;  3248          3366  1                 VECTOR[DBG$K_MAX_LANGUAGE + 1, LONG] PSECT(DBG$PLIT) PRESET(
;  3249          3367  1
;  3250          3368  1         [DBG$K_MACRO]   = MACRO_TABLES   - TABLEBASE,    ! MACRO
;  3251          3369  1         [DBG$K_FORTRAN] = FORTRAN_TABLES - TABLEBASE,    ! FORTRAN
;  3252          3370  1         [DBG$K_BLISS]   = BLISS_TABLES   - TABLEBASE,    ! BLISS
;  3253          3371  1         [DBG$K_COBOL]   = COBOL_TABLES   - TABLEBASE,    ! COBOL
;  3254          3372  1         [DBG$K_BASIC]   = BASIC_TABLES   - TABLEBASE,    ! BASIC
;  3255          3373  1         [DBG$K_PLI]     = PLI_TABLES     - TABLEBASE,    ! PL/I
;  3256          3374  1         [DBG$K_PASCAL]  = PASCAL_TABLES  - TABLEBASE,    ! PASCAL
;  3257          3375  1         [DBG$K_C]       = C_TABLES       - TABLEBASE,    ! C
;  3258          3376  1         [DBG$K_RPG]     = RPG_TABLES     - TABLEBASE,    ! RPG
;  3259          3377  1         [DBG$K_ADA]     = ADA_TABLES     - TABLEBASE,    ! ADA
;  3260          3378  1         [DBG$K_UNKNOWN] = UNKNOWN_TABLES - TABLEBASE);   ! UNKNOWN
```

```
;  3262          3379  1 ROUTINE AAA_DUMMY: NOVALUE =
;  3263          3380  1 !
;  3264          3381  1 ! FUNCTION
;  3265          3382  1 !       This is a dummy routine which is never called.  Its sole purpose is
;  3266          3383  1 !       to force out the machine listing for all the above data tables before
;  3267          3384  1 !       the source listing for the first real routine in the module.
;  3268          3385  1 !
;  3269          3386  1 ! INPUTS
;  3270          3387  1 !       NONE
;  3271          3388  1 !
;  3272          3389  1 ! OUTPUTS
;  3273          3390  1 !       NONE
;  3274          3391  1 !
;  3275          3392  1
;  3276          3393  2     BEGIN
;  3277          3394  2
;  3278          3395  2     RETURN;
;  3279          3396  2
;  3280          3397  1     END;

                                                        .TITLE  DBGPARSER
                                                        IDENT  \V04-000\

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

                                        00  03  00000 P.AAA:  .BYTE   3, 0                                                        ;
                                          0033  00002         .WORD   51                                                         ;
                    00  00  00  00  00  00  00  00  00004     .BYTE   0, 0, 0, 0, 0, 0, 0, 0                                     ;
                    54  52  45  56  4E  4F  43  07  0000C     .ASCII  <7>\CONVERT\
                                        00  03  00014 P.AAB:  .BYTE   3, 0
                                          0032  00016         .WORD   50
                    00  00  00  00  00  00  00  00  00018     .BYTE   0, 0, 0, 0, 0, 0, 0, 0                                     ;
                    54  49  53  4F  50  45  44  07  00020     .ASCII  <7>\DEPOSIT\
                                        00  02  00028 P.AAC:  .BYTE   2, 0
                                          0038  0002A         .WORD   56
                    00  00  00  00  00  00  00  00  0002C     .BYTE   0, 0, 0, 0, 0, 0, 0, 0                                     ;
                59  54  49  54  4E  45  44  49  08  00034     .ASCII  <8>\IDENTITY\
                                        00  02  0003D P.AAD:  .BYTE   2, 0
                                          0042  0003F         .WORD   66
                    00  00  00  00  00  00  00  00  00041     .BYTE   0, 0, 0, 0, 0, 0, 0, 0                                     ;
                54  53  4E  4F  43  47  45  4E  08  00049     .ASCII  <8>\NEGCONST\
                                        00  02  00052 P.AAE:  .BYTE   2, 0
                                          0043  00054         .WORD   67
                    00  00  00  00  00  00  00  00  00056     .BYTE   0, 0, 0, 0, 0, 0, 0, 0                                     ;
                54  53  4E  4F  43  53  4F  50  08  0005E     .ASCII  <8>\POSCONST\
                                        00  02  00067 P.AAF:  .BYTE   2, 0
                                          0005  00069         .WORD   5
                    00  00  00  00  00  00  00  00  0006B     .BYTE   0, 0, 0, 0, 0, 0, 0, 0                                     ;
                                        2D  01  00073         .ASCII  <1>\-\
                                        00  02  00075 P.AAG:  .BYTE   2, 0
                                          0004  00077         .WORD   4
                    00  00  00  00  00  00  00  00  00079     .BYTE   0, 0, 0, 0, 0, 0, 0, 0                                     ;
                                        2B  01  00081         .ASCII  <1>\+\
                                    45  53  41  42  00083 P.AAH:  .ASCII  \BASE\
                                            01  00087 P.AAJ:  .BYTE   1
                        45  4E  49  4C  25  05  00088         .ASCII  <5>\%LINE\                                                 ;
```

```
                                                01  0008E P.AAK:   .BYTE    1
                                            4E  49  4C  25  04  0008F          .ASCII   <4>\%LIN\
                                                01  00094 P.AAL:   .BYTE    1
                                                49  4C  25  03  00095          .ASCII   <3>\%LI\
                                                02  00099 P.AAM:   .BYTE    2
                                        4C  45  42  41  4C  25  06  0009A      .ASCII   <6>\%LABEL\
                                                02  000A1 P.AAN:   .BYTE    2
                                            45  42  41  4C  25  05  000A2      .ASCII   <5>\%LABE\
                                                02  000A8 P.AAO:   .BYTE    2
                                                42  41  4C  25  04  000A9      .ASCII   <4>\%LAB\
                                                02  000AE P.AAP:   .BYTE    2
                                                    41  4C  25  03  000AF      .ASCII   <3>\%LA\
                                                03  000B3 P.AAQ:   .BYTE    3
                                            45  4D  41  4E  25  05  000B4      .ASCII   <5>\%NAME\
                                                08  000BA P.AAR:   .BYTE    8
                                                    30  52  25  03  000BB      .ASCII   <3>\%R0\
                                                08  000BF P.AAS:   .BYTE    8
                                                    31  52  25  03  000C0      .ASCII   <3>\%R1\
                                                08  000C4 P.AAT:   .BYTE    8
                                                    32  52  25  03  000C5      .ASCII   <3>\%R2\
                                                08  000C9 P.AAU:   .BYTE    8
                                                    33  52  25  03  000CA      .ASCII   <3>\%R3\
                                                08  000CE P.AAV:   .BYTE    8
                                                    34  52  25  03  000CF      .ASCII   <3>\%R4\
                                                08  000D3 P.AAW:   .BYTE    8
                                                    35  52  25  03  000D4      .ASCII   <3>\%R5\
                                                08  000D8 P.AAX:   .BYTE    8
                                                    36  52  25  03  000D9      .ASCII   <3>\%R6\
                                                08  000DD P.AAY:   .BYTE    8
                                                    37  52  25  03  000DE      .ASCII   <3>\%R7\
                                                08  000E2 P.AAZ:   .BYTE    8
                                                    38  52  25  03  000E3      .ASCII   <3>\%R8\
                                                08  000E7 P.ABA:   .BYTE    8
                                                    39  52  25  03  000E8      .ASCII   <3>\%R9\
                                                08  000EC P.ABB:   .BYTE    8
                                        30  31  52  25  04  000ED          .ASCII   <4>\%R10\
                                                08  000F2 P.ABC:   .BYTE    8
                                        31  31  52  25  04  000F3          .ASCII   <4>\%R11\
                                                08  000F8 P.ABD:   .BYTE    8
                                        32  31  52  25  04  000F9          .ASCII   <4>\%R12\
                                                08  000FE P.ABE:   .BYTE    8
                                        33  31  52  25  04  000FF          .ASCII   <4>\%R13\
                                                08  00104 P.ABF:   .BYTE    8
                                        34  31  52  25  04  00105          .ASCII   <4>\%R14\
                                                08  0010A P.ABG:   .BYTE    8
                                        35  31  52  25  04  0010B          .ASCII   <4>\%R15\
                                                08  00110 P.ABH:   .BYTE    8
                                                    50  41  25  03  00111      .ASCII   <3>\%AP\
                                                08  00115 P.ABI:   .BYTE    8
                                                    50  46  25  03  00116      .ASCII   <3>\%FP\
                                                08  0011A P.ABJ:   .BYTE    8
                                                    50  53  25  03  0011B      .ASCII   <3>\%SP\
                                                08  0011F P.ABK:   .BYTE    8
                                                    43  50  25  03  00120      .ASCII   <3>\%PC\
                                                08  00124 P.ABL:   .BYTE    8
                                        4C  53  50  25  04  00125          .ASCII   <4>\%PSL\
                                                08  0012A P.ABM:   .BYTE    8
```

```
                30  72  25  03  0012B          .ASCII  <3>\%r0\
                                08  0012F P.ABN:  .BYTE   8
                31  72  25  03  00130          .ASCII  <3>\%r1\
                                08  00134 P.ABO:  .BYTE   8
                32  72  25  03  00135          .ASCII  <3>\%r2\
                                08  00139 P.ABP:  .BYTE   8
                33  72  25  03  0013A          .ASCII  <3>\%r3\
                                08  0013E P.ABQ:  .BYTE   8
                34  72  25  03  0013F          .ASCII  <3>\%r4\
                                08  00143 P.ABR:  .BYTE   8
                35  72  25  03  00144          .ASCII  <3>\%r5\
                                08  00148 P.ABS:  .BYTE   8
                36  72  25  03  00149          .ASCII  <3>\%r6\
                                08  0014D P.ABT:  .BYTE   8
                37  72  25  03  0014E          .ASCII  <3>\%r7\
                                08  00152 P.ABU:  .BYTE   8
                38  72  25  03  00153          .ASCII  <3>\%r8\
                                08  00157 P.ABV:  .BYTE   8
                39  72  25  03  00158          .ASCII  <3>\%r9\
                                08  0015C P.ABW:  .BYTE   8
            30  31  72  25  04  0015D          .ASCII  <4>\%r10\
                                08  00162 P.ABX:  .BYTE   8
            31  31  72  25  04  00163          .ASCII  <4>\%r11\
                                08  00168 P.ABY:  .BYTE   8
            32  31  72  25  04  00169          .ASCII  <4>\%r12\
                                08  0016E P.ABZ:  .BYTE   8
            33  31  72  25  04  0016F          .ASCII  <4>\%r13\
                                08  00174 P.ACA:  .BYTE   8
            34  31  72  25  04  00175          .ASCII  <4>\%r14\
                                08  0017A P.ACB:  .BYTE   8
            35  31  72  25  04  0017B          .ASCII  <4>\%r15\
                                08  00180 P.ACC:  .BYTE   8
                70  61  25  03  00181          .ASCII  <3>\%ap\
                                08  00185 P.ACD:  .BYTE   8
                70  66  25  03  00186          .ASCII  <3>\%fp\
                                08  0018A P.ACE:  .BYTE   8
                70  73  25  03  0018B          .ASCII  <3>\%sp\
                                08  0018F P.ACF:  .BYTE   8
                63  70  25  03  00190          .ASCII  <3>\%pc\
                                08  00194 P.ACG:  .BYTE   8
            6C  73  70  25  04  00195          .ASCII  <4>\%psl\
                                04  0019A P.ACH:  .BYTE   4
            43  45  44  25  04  0019B          .ASCII  <4>\%DEC\
                                05  001A0 P.ACI:  .BYTE   5
            58  45  48  25  04  001A1          .ASCII  <4>\%HEX\
                                06  001A6 P.ACJ:  .BYTE   6
            54  43  4F  25  04  001A7          .ASCII  <4>\%OCT\
                                07  001AC P.ACK:  .BYTE   7
            4E  49  42  25  04  001AD          .ASCII  <4>\%BIN\
                                08  001B2 P.ACL:  .BYTE   8
        43  4F  4C  52  55  43  25  07  001B3          .ASCII  <7>\%CURLOC\
                                08  001BB P.ACM:  .BYTE   8
        4C  41  56  52  55  43  25  07  001BC          .ASCII  <7>\%CURVAL\
                                08  001C4 P.ACN:  .BYTE   8
43  4F  4C  56  45  52  50  25  08  001C5          .ASCII  <8>\%PREVLOC\
                                08  001CE P.ACO:  .BYTE   8
43  4F  4C  54  58  45  4E  25  08  001CF          .ASCII  <8>\%NEXTLOC\
```

```
                                                                        08  001D8 P.ACP:   .BYTE   8
                                      54  4E  43  52  41  50  25  07  001D9           .ASCII  <7>\%PARCNT\
                                                                         001E1           .BLKB   3
                                                                  0000003B  001E4           .LONG   59
00000025  0000001E  00000016  00000011  0000000B  00000004  001E8 P.AAI:   .LONG   4, 11, 17, 22, 30, 37, 43, 48, 55, 60, -
00000046  00000041  0000003C  00000037  00000030  0000002B  00200                   65, 70, 75, 80, 85, 90, 95, 100, 105, -
00000064  0000005F  0000005A  00000055  00000050  0000004B  00218                   111, 117, 123, 129, 135, 141, 146, 151, -
00000087  00000081  0000007B  00000075  0000006F  00000069  00230                   156, 161, 167, 172, 177, 182, 187, 192, -
000000A7  000000A1  0000009C  00000097  00000092  0000008D  00248                   197, 202, 207, 212, 217, 223, 229, 235, -
000000C5  000000C0  000000BB  000000B6  000000B1  000000AC  00260                   241, 247, 253, 258, 263, 268, 273, 279, -
000000E5  000000DF  000000D9  000000D4  000000CF  000000CA  00278                   285, 291, 297, 303, 312, 321, 331, 341
00000107  00000102  000000FD  000000F7  000000F1  000000EB  00290
00000129  00000123  0000011D  00000117  00000111  0000010C  002A8
          00000155  0000014B  00000141  00000138  0000012F  002C0
                                                          00  02  002D4 P.ACQ:   .BYTE   2, 0
                                                            0001  002D6           .WORD   1
                                  00  00  00  00  00  00  C8  01  002D8           .BYTE   1, -56, 0, 0, 0, 0, 0, 0
65  72  70  78  65  20  66  6F  20  74  72  61  74  73  13  002E0           .ASCII  <19>\start of expression\
                                              6E  6F  69  73  73  002EF
                                                          02  04  002F4 P.ACR:   .BYTE   4, 2
                                                            0002  002F6           .WORD   2
                                  00  00  00  00  00  00  02  C8  002F8           .BYTE   -56, 2, 0, 0, 0, 0, 0, 0
73  73  65  72  70  78  65  20  66  6F  20  64  6E  65  11  00300           .ASCII  <17>\end of expression\
                                                      6E  6F  69  0030F
                                                          01  04  00312 P.ACS:   .BYTE   4, 1
                                                            0001  00314           .WORD   1
                                  00  00  00  00  00  00  00  00  00316           .BYTE   0, 0, 0, 0, 0, 0, 0, 0
    6C  6F  62  6D  79  73  20  66  6F  20  64  6E  65  0D  0031E           .ASCII  <13>\end of symbol\
                                                          02  02  0032C P.ACT:   .BYTE   2, 2
                                                            0034  0032E           .WORD   52
                                  00  00  00  00  00  00  C8  BE  00330           .BYTE   -66, -56, 0, 0, 0, 0, 0, 0
                                              43  45  44  25  04  00338           .ASCII  <4>\%DEC\
                                                          02  02  0033D P.ACU:   .BYTE   2, 2
                                                            0035  0033F           .WORD   53
                                  00  00  00  00  00  00  C8  BE  00341           .BYTE   -66, -56, 0, 0, 0, 0, 0, 0
                                              58  45  48  25  04  00349           .ASCII  <4>\%HEX\
                                                          02  02  0034E P.ACV:   .BYTE   2, 2
                                                            0036  00350           .WORD   54
                                  00  00  00  00  00  00  C8  BE  00352           .BYTE   -66, -56, 0, 0, 0, 0, 0, 0
                                              54  43  4F  25  04  0035A           .ASCII  <4>\%OCT\
                                                          02  02  0035F P.ACW:   .BYTE   2, 2
                                                            0037  00361           .WORD   55
                                  00  00  00  00  00  00  C8  BE  00363           .BYTE   -66, -56, 0, 0, 0, 0, 0, 0
                                              4E  49  42  25  04  0036B           .ASCII  <4>\%BIN\
                                                          00  01  00370 P.ACX:   .BYTE   1, 0
                                                            0001  00372           .WORD   1
                                                  00  00  00  00  00374           .BYTE   0, 0, 0, 0
                                  43  4F  4C  52  55  43  25  07  00378           .ASCII  <7>\%CURLOC\
                                                          00  01  00380 P.ACY:   .BYTE   1, 0
                                                            0001  00382           .WORD   1
                                                  00  00  00  00  00384           .BYTE   0, 0, 0, 0
                                  4C  41  56  52  55  43  25  07  00388           .ASCII  <7>\%CURVAL\
                                                          00  01  00390 P.ACZ:   .BYTE   1, 0
                                                            0001  00392           .WORD   1
                                                  00  00  00  00  00394           .BYTE   0, 0, 0, 0
                              43  4F  4C  56  45  52  50  25  08  00398           .ASCII  <8>\%PREVLOC\
                                                          00  02  003A1 P.ADB:   .BYTE   2, 0
```

```
                                           0003  003A3          .WORD   3                                         :
                00  00  00  00  00  00  C8   28  003A5          .BYTE   40, -56, 0, 0, 0, 0, 0, 0                 :
                                        2E   01  003AD          .ASCII  <1>\.\                                    :
                                        00   02  003AF P.ADC:   .BYTE   2, 0                                      :
                                           0003  003B1          .WORD   3                                         :
                00  00  00  00  00  00  C8   28  003B3          .BYTE   40, -56, 0, 0, 0, 0, 0, 0                 :
                                        40   01  003BB          .ASCII  <1>\@\                                    :
                                        00   03  003BD P.ADD:   .BYTE   3, 0                                      :
                                           0006  003BF          .WORD   6                                         :
                00  00  00  00  00  00  0A   0A  003C1          .BYTE   10, 10, 0, 0, 0, 0, 0, 0                  :
                                        2B   01  003C9          .ASCII  <1>\+\                                    :
                                        00   03  003CB P.ADE:   .BYTE   3, 0                                      :
                                           0007  003CD          .WORD   7                                         :
                00  00  00  00  00  00  0A   0A  003CF          .BYTE   10, 10, 0, 0, 0, 0, 0, 0                  :
                                        2D   01  003D7          .ASCII  <1>\-\                                    :
                                        00   02  003D9 P.ADF:   .BYTE   2, 0                                      :
                                           0004  003DB          .WORD   4                                         :
                00  00  00  00  00  00  C8   14  003DD          .BYTE   20, -56, 0, 0, 0, 0, 0, 0                 :
                                        2B   01  003E5          .ASCII  <1>\+\                                    :
                                        00   02  003E7 P.ADG:   .BYTE   2, 0                                      :
                                           0005  003E9          .WORD   5                                         :
                00  00  00  00  00  00  C8   14  003EB          .BYTE   20, -56, 0, 0, 0, 0, 0, 0                 :
                                        2D   01  003F3          .ASCII  <1>\-\                                    :
                                        00   03  003F5 P.ADH:   .BYTE   3, 0                                      :
                                           0008  003F7          .WORD   8                                         :
                00  00  00  00  00  00  1E   1E  003F9          .BYTE   30, 30, 0, 0, 0, 0, 0, 0                  :
                                        2A   01  00401          .ASCII  <1>\*\                                    :
                                        00   03  00403 P.ADI:   .BYTE   3, 0                                      :
                                           0009  00405          .WORD   9                                         :
                00  00  00  00  00  00  1E   1E  00407          .BYTE   30, 30, 0, 0, 0, 0, 0, 0                  :
                                        2F   01  0040F          .ASCII  <1>\/\                                    :
                                        02   04  00411 P.ADJ:   .BYTE   4, 2                                      :
                                           0031  00413          .WORD   49                                        :
                00  00  00  00  00  00  32   C8  00415          .BYTE   -56, 50, 0, 0, 0, 0, 0, 0                 :
                                        3C   01  0041D          .ASCII  <1>\<\                                    :
                                        02   02  0041F P.ADK:   .BYTE   2, 2                                      :
                                           000B  00421          .WORD   11                                        :
                00  00  00  00  00  00  C8   05  00423          .BYTE   5, -56, 0, 0, 0, 0, 0, 0                  :
                                        28   01  0042B          .ASCII  <1>\(\                                    :
                                        02   04  0042D P.ADL:   .BYTE   4, 2                                      :
                                           000C  0042F          .WORD   12                                        :
                00  00  00  00  00  00  06   C8  00431          .BYTE   -56, 6, 0, 0, 0, 0, 0, 0                  :
                                        29   01  00439          .ASCII  <1>\)\                                    :
                                                 0043B          .BLKB   1                                         :
                                       0000000B  0043C          .LONG   11                                        :
00000364  00000356  00000348  0000033A  0000032C  0000031E  00440 P.ADA:   .LONG   798, 812, 826, 840, 854, 868, 882, 896, -  :
          000003AA  0000039C  0000038E  00000380  00000372  00458                  910, 924, 938                              :
                                       00000000  0046C          .LONG   0                                         :
                                                 00470 P.ADM:   .BLKB   0
                                        00   00  00470 P.ADO:   .BYTE   0, 0                                      :
                                           0001  00472          .WORD   1                                         :
                                00  00  00   00  00474          .BYTE   0, 0, 0, 0                                :
                                        2C   01  00478          .ASCII  <1>\,\                                    :
                                                 0047A          .BLKB   2                                         :
                                       00000001  0047C          .LONG   1                                         :
                                       000003ED  00480 P.ADN:   .LONG   1005                                      :
                                        01   00  00484 P.ADQ:   .BYTE   0, 1                                      :
```

```
                          0004  00486          .WORD   4                       ;
                00  00    00  00  00488          .BYTE   0, 0, 0, 0              ;
                          3D  01  0048C          .ASCII  <1>\=\                  ;
                              00  0048E          .BLKB   2                       ;
                      00000001  00490          .LONG   1                       ;
                      00000401  00494 P.ADP:   .LONG   1025                    ;
                        01  00  00498 P.ADS:   .BYTE   0, 1                     ;
                          0005  0049A          .WORD   5                       ;
                00  00  00  00  0049C          .BYTE   0, 0, 0, 0              ;
                    4F  44  02  004A0          .ASCII  <2>\DO\                 ;
                              004A3          .BLKB   1                       ;
                      00000001  004A4          .LONG   1                       ;
                      00000415  004A8 P.ADR:   .LONG   1045                    ;
                        01  00  004AC P.ADU:   .BYTE   0, 1                     ;
                          0006  004AE          .WORD   6                       ;
                00  00  00  00  004B0          .BYTE   0, 0, 0, 0              ;
            4E  45  48  54  04  004B4          .ASCII  <4>\THEN\               ;
                              004B9          .BLKB   3                       ;
                      00000001  004BC          .LONG   1                       ;
                      00000429  004C0 P.ADT:   .LONG   1065                    ;
                        00  00  004C4 P.ADW:   .BYTE   0, 0                     ;
                          0001  004C6          .WORD   1                       ;
                00  00  00  00  004C8          .BYTE   0, 0, 0, 0              ;
                          2C  01  004CC          .ASCII  <1>\,\                  ;
                          02  00  004CE P.ADX:   .BYTE   0, 2                     ;
                          0003  004D0          .WORD   3                       ;
                C0  00  00  00  004D2          .BYTE   0, 0, 0, 0              ;
                          3A  01  004D6          .ASCII  <1>\:\                  ;
                      00000002  004D8          .LONG   2                       ;
        0000044B      00000441  004DC P.ADV:   .LONG   1089, 1099              ;
                        00  00  004E4 P.ADZ:   .BYTE   0, 0                     ;
                          0001  004E6          .WORD   1                       ;
                00  00  00  00  004E8          .BYTE   0, 0, 0, 0              ;
                          2C  01  004EC          .ASCII  <1>\,\                  ;
                          01  00  004EE P.AEA:   .BYTE   0, 1                     ;
                          0007  004F0          .WORD   7                       ;
                00  00  00  00  004F2          .BYTE   0, 0, 0, 0              ;
            4E  45  48  57  04  004F6          .ASCII  <4>\WHEN\               ;
                          01  00  004FB P.AEB:   .BYTE   0, 1                     ;
                          0005  004FD          .WORD   5                       ;
                00  00  00  00  004FF          .BYTE   0, 0, 0, 0              ;
                    4F  44  02  00503          .ASCII  <2>\DO\                 ;
                              00506          .BLKB   2                       ;
                      00000003  00508          .LONG   3                       ;
00000478        0000046B      00000461  0050C P.ADY:   .LONG   1121, 1131, 1144;
                        00  00  00518 P.AED:   .BYTE   0, 0                     ;
                          000B  0051A          .WORD   11                      ;
                00  00  00  00  0051C          .BYTE   0, 0, 0, 0              ;
                          28  01  00520          .ASCII  <1>\(\                  ;
                              00522          .BLKB   2                       ;
                      00000001  00524          .LONG   1                       ;
                      00000495  00528 P.AEC:   .LONG   1173                    ;
                        00  00  0052C P.AEF:   .BYTE   0, 0                     ;
                          0001  0052E          .WORD   1                       ;
                00  00  00  00  00530          .BYTE   0, 0, 0, 0              ;
                          2C  01  00534          .ASCII  <1>\,\                  ;
                          01  00  00536 P.AEG:   .BYTE   0, 1                     ;
```

```
                    0002  00538          .WORD   2
    00  00  00  00   0053A          .BYTE   0, 0, 0, 0
            29  01   0053E          .ASCII  <1>\)\
              00000002  00540          .LONG   2
000004B3      000004A9  00544 P.AEE:   .LONG   1193, 1203
              01  00    0054C P.AEI:   .BYTE   0, 1
                000D    0054E          .WORD   13
    00  00  00  00   00550          .BYTE   0, 0, 0, 0
        4F  54  02   00554          .ASCII  <2>\TO\
                     00557          .BLKB   1
              00000001  00558          .LONG   1
              000004C9  0055C P.AEH:   .LONG   1225
              01  00    00560 P.AEK:   .BYTE   0, 1
                000E    00562          .WORD   14
    00  00  00  00   00564          .BYTE   0, 0, 0, 0
        59  42  02   00568          .ASCII  <2>\BY\
              01  00    0056B P.AEL:   .BYTE   0, 1
                0005    0056D          .WORD   5
    00  00  00  00   0056F          .BYTE   0, 0, 0, 0
        4F  44  02   00573          .ASCII  <2>\DO\
                     00576          .BLKB   2
              00000002  00578          .LONG   2
000004E8      000004DD  0057C P.AEJ:   .LONG   1245, 1256
              00  00    00584 P.AEN:   .BYTE   0, 0
                0001    00586          .WORD   1
    00  00  00  00   00588          .BYTE   0, 0, 0, 0
              2C  01   0058C          .ASCII  <1>\,\
              00  00    0058E P.AEO:   .BYTE   0, 0
                000A    00590          .WORD   10
    00  00  00  00   00592          .BYTE   0, 0, 0, 0
              3E  01   00596          .ASCII  <1>\>\
              00000002  00598          .LONG   2
          0000050B  00000501  0059C P.AEM:   .LONG   1281, 1291
                00  00    005A4 P.AEQ:   .BYTE   0, 0
                  0001    005A6          .WORD   1
        00  00  00  00   005A8          .BYTE   0, 0, 0, 0
                2C  01   005AC          .ASCII  <1>\,\
                01  00    005AE P.AER:   .BYTE   0, 1
                  0002    005B0          .WORD   2
        00  00  00  00   005B2          .BYTE   0, 0, 0, 0
                5D  01   005B6          .ASCII  <1>\]\
                00  00    005B8 P.AES:   .BYTE   0, 0
                  000F    005BA          .WORD   15
        00  00  00  00   005BC          .BYTE   0, 0, 0, 0
            2E  2E  02   005C0          .ASCII  <2>\..\
                     005C3          .BLKB   1
              00000003  005C4          .LONG   3
  00000535    0000052B  00000521  005C8 P.AEP:   .LONG   1313, 1323, 1333
              000003FD  000003ED  005D4 TERM_POINTER_TBL:
                                         .LONG   1005, 1021
                        00#      005DC          .BYTE   0[8]
  0000043D    00000425  00000411  005E4          .LONG   1041, 1061, 1085
                        00#      005F0          .BYTE   0[4]
              00000489  00000459  005F4          .LONG   1113, 1161
                        00#      005FC          .BYTE   0[4]
000004F9      000004D9  000004C1  000004A5  00600          .LONG   1189, 1217, 1241, 1273
                                  00610          .BLKB   4
```

```
                                                        00# 00614 BASE_CHARACTER_TABLE:
                                                                   .BYTE   0[36]
                                                00020000  00638    .LONG   131072
                                                      00# 0063C    .BYTE   0[88]
00000000  00000006  00000000  00000400  00000000  00020000  00694    .LONG   131072, 0, 1024, 0, 6, 0, 0, 1024, -
00008840  0000A800  00048800  00048800  00000400  00000000  006AC            296960, 296960, 43008, 34880, 262144, -
0000021E  0000021E  00008800  00018938  00008850  00040000  006C4            34896, 100408, 34816, 542, 542, 542, 542, -
0000021E  0000021E  0000021E  0000021E  0000021E  0000021E  006DC            542, 542, 542, 542, 542, 542, 262144, 0, -
00040000  00008000  00000000  00040000  0000021E  0000021E  006F4            32768, 262144, 262144, 0, 34816, 295, -
00000127  00040167  00000127  00008800  00000000  00040000  0070C            262503, 295, 262519, 391, 295, 471, 263, -
00000107  00000107  000001D7  00000127  00000187  00040177  00724            263, 263, 263, 263, 263, 263, 263, 263, -
00000107  00000107  00000107  00000107  00000107  00000107  0073C            407, 263, 263, 262407, 263, 263, 262407, -
00000107  00040107  00000107  00000107  00000197  00000107  00754            423, 263, 263, 0, 67584, 0, 65536, 6, 0, -
00000000  00000107  00000107  000001A7  00040107  00000107  0076C            295, 359, 295, 262519, 391, 295, 471, -
00000127  00000000  00000006  00010000  00000000  00010800  00784            263, 263, 263, 263, 263, 263, 263, 263, -
000001D7  00000127  00000187  00040177  00000127  00000167  0079C            263, 407, 263, 263, 262407, 263, 263, -
00000107  00000107  00000107  00000107  00000107  00000107  007B4            262407, 423, 263, 263, 0, 0, 0, 0
00000107  00000107  00000197  00000107  00000107  00000107  007CC
00000107  000001A7  00040107  00000107  00000107  00040107  007E4
          00000000  00000000  00000000  00000000  00000107  007FC
                                                      00810    .BLKB   516
                                                00000008  00A14    .LONG   8
5E010800  3E042800  3D042800  3C00A800  2F00A800  26000800  00A18 P.AET: .LONG   637536256, 788572160, 1006675968, -
                                        5D040000  5B000800  00A30            1023682560, 1040459776, 1577125888, -
                                                                             1526728704, 1560543232
                                                    00  03  00A38 P.AEV: .BYTE   3, 0
                                                      000D  00A3A          .WORD   13
                                00  00  00  00  00  00  32  32  00A3C          .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                            4C  51  45  03  00A44          .ASCII  <3>\EQL\
                                                    00  03  00A48 P.AEW: .BYTE   3, 0
                                                      000E  00A4A          .WORD   14
00  00  00  00  00  00  32  32  00A4C          .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                            51  45  4E  03  00A54          .ASCII  <3>\NEQ\
                                                    00  03  00A58 P.AEX: .BYTE   3, 0
                                                      000F  00A5A          .WORD   15
00  00  00  00  00  00  32  32  00A5C          .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                            52  54  47  03  00A64          .ASCII  <3>\GTR\
                                                    00  03  00A68 P.AEY: .BYTE   3, 0
                                                      0011  00A6A          .WORD   17
00  00  00  00  00  00  32  32  00A6C          .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                            51  45  47  03  00A74          .ASCII  <3>\GEQ\
                                                    00  03  00A78 P.AEZ: .BYTE   3, 0
                                                      0013  00A7A          .WORD   19
00  00  00  00  00  00  32  32  00A7C          .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                            53  53  4C  03  00A84          .ASCII  <3>\LSS\
                                                    00  03  00A88 P.AFA: .BYTE   3, 0
                                                      0015  00A8A          .WORD   21
00  00  00  00  00  00  32  32  00A8C          .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                            51  45  4C  03  00A94          .ASCII  <3>\LEQ\
                                                    00  02  00A98 P.AFS: .BYTE   2, 0
                                                      0017  00A9A          .WORD   23
00  00  00  00  00  00  C8  28  00A9C          .BYTE   40, -56, 0, 0, 0, 0, 0, 0
                            54  4F  4E  03  00AA4          .ASCII  <3>\NOT\
                                                    00  03  00AA8 P.AFC: .BYTE   3, 0
                                                      0018  00AAA          .WORD   24
00  00  00  00  00  00  1E  1E  00AAC          .BYTE   30, 30, 0, 0, 0, 0, 0, 0
                            44  4E  41  03  00AB4          .ASCII  <3>\AND\
```

```
                                                    00  03   00AB8 P.AFD:   .BYTE   3, 0
                                                      0019   00ABA          .WORD   25
                                00  00  00  00  00  00  14  14   00ABC          .BYTE   20, 20, 0, 0, 0, 0, 0, 0
                                                        52  4F  02   00AC4          .ASCII  <2>\OR\
                                                    00  03   00AC7 P.AFE:   .BYTE   3, 0
                                                      001A   00AC9          .WORD   26
                                00  00  00  00  00  00  0A  0A   00ACB          .BYTE   10, 10, 0, 0, 0, 0, 0, 0
                                                        52  4F  58  03   00AD3          .ASCII  <3>\XOR\
                                                    00  03   00AD7 P.AFF:   .BYTE   3, 0
                                                      001B   00AD9          .WORD   27
                                00  00  00  00  00  00  0A  0A   00ADB          .BYTE   10, 10, 0, 0, 0, 0, 0, 0
                                                        56  51  45  03   00AE3          .ASCII  <3>\EQV\
                                                               00AE7          .BLKB   1
                                                  0000000B   00AE8          .LONG   11
00000A05  000009F5  000009E5  000009D5  000009C5  000009B5   00AEC P.AEU:   .LONG   2485, 2501, 2517, 2533, 2549, 2565, 2581, -
          00000A54  00000A44  00000A35  00000A25  00000A15   00B04                  2597, 2613, 2628, 2644
                                                    01  02   00B18 P.AFH:   .BYTE   2, 1
                                                      0002   00B1A          .WORD   2
                                00  00  00  00  00  00  00  00   00B1C          .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                    5C  01   00B24          .ASCII  <1><92>
                                                    01  03   00B26 P.AFI:   .BYTE   3, 1
                                                      0003   00B28          .WORD   3
                                00  00  00  00  00  00  00  00   00B2A          .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                    5C  01   00B32          .ASCII  <1><92>
                                                    01  03   00B34 P.AFJ:   .BYTE   3, 1
                                                      0005   00B36          .WORD   5
                                00  00  00  00  00  00  00  00   00B38          .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                    2E  01   00B40          .ASCII  <1>\.\
                                                    01  04   00B42 P.AFK:   .BYTE   4, 1
                                                      0004   00B44          .WORD   4
00  00  00  00  00  00  00  00   00B46          .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                    28  01   00B4E          .ASCII  <1>\(\
                                                    01  04   00B50 P.AFL:   .BYTE   4, 1
                                                      0004   00B52          .WORD   4
00  00  00  00  00  00  00  00   00B54          .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                    5B  01   00B5C          .ASCII  <1>\[\
                                                    01  04   00B5E P.AFM:   .BYTE   4, 1
                                                      0007   00B60          .WORD   7
00  00  00  00  00  00  00  00   00B62          .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                    5E  01   00B6A          .ASCII  <1>\^\
                                                    00  03   00B6C P.AFN:   .BYTE   3, 0
                                                      0023   00B6E          .WORD   35
00  00  00  00  00  00  3C  3C   00B70          .BYTE   60, 60, 0, 0, 0, 0, 0, 0
                                                        2F  2F  02   00B78          .ASCII  <2>\//\
                                                    00  03   00B7B P.AFO:   .BYTE   3, 0
                                                      0023   00B7D          .WORD   35
00  00  00  00  00  00  3C  3C   00B7F          .BYTE   60, 60, 0, 0, 0, 0, 0, 0
                                                    26  01   00B87          .ASCII  <1>\&\
                                                    00  03   00B89 P.AFP:   .BYTE   3, 0
                                                      0006   00B8B          .WORD   6
00  00  00  00  00  00  3C  3C   00B8D          .BYTE   60, 60, 0, 0, 0, 0, 0, 0
                                                    2B  01   00B95          .ASCII  <1>\+\
                                                    00  03   00B97 P.AFQ:   .BYTE   3, 0
                                                      0007   00B99          .WORD   7
00  00  00  00  00  00  3C  3C   00B9B          .BYTE   60, 60, 0, 0, 0, 0, 0, 0
                                                    2D  01   00BA3          .ASCII  <1>\-\
                                                    00  02   00BA5 P.AFR:   .BYTE   2, 0
```

```
                        0004 00BA7           .WORD   4
00  00  00  00  00  00  C8  46 00BA9           .BYTE   70, -56, 0, 0, 0, 0, 0, 0
                        2B  01 00BB1           .ASCII  <1>\+\
                        00  02 00BB3 P.AFS:    .BYTE   2, 0
                        0005   00BB5           .WORD   5
00  00  00  00  00  00  C8  46 00BB7           .BYTE   70, -56, 0, 0, 0, 0, 0, 0
                        2D  01 00BBF           .ASCII  <1>\-\
                        00  03 00BC1 P.AFT:    .BYTE   3, 0
                        0008   00BC3           .WORD   8
00  00  00  00  00  00  50  50 00BC5           .BYTE   80, 80, 0, 0, 0, 0, 0, 0
                        2A  01 00BCD           .ASCII  <1>\*\
                        00  03 00BCF P.AFU:    .BYTE   3, 0
                        0009   00BD1           .WORD   9
00  00  00  00  00  00  50  50 00BD3           .BYTE   80, 80, 0, 0, 0, 0, 0, 0
                        2F  01 00BDB           .ASCII  <1>\/\
                        00  03 00BDD P.AFV:    .BYTE   3, 0
                        000A   00BDF           .WORD   10
00  00  00  00  00  00  5C  5A 00BE1           .BYTE   90, 92, 0, 0, 0, 0, 0, 0
                    2A  2A  02 00BE9           .ASCII  <2>\**\
                        00  03 00BEC P.AFW:    .BYTE   3, 0
                        000D   00BEE           .WORD   13
00  00  00  00  00  00  32  32 00BF0           .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                        3D  01 00BF8           .ASCII  <1>\=\
                        00  03 00BFA P.AFX:    .BYTE   3, 0
                        000E   00BFC           .WORD   14
00  00  00  00  00  00  32  32 00BFE           .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                    3E  3C  02 00C06           .ASCII  <2>\<>\
                        00  03 00C09 P.AFY:    .BYTE   3, 0
                        000E   00C0B           .WORD   14
00  00  00  00  00  00  32  32 00C0D           .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                    3D  2F  02 00C15           .ASCII  <2>\/=\
                        00  03 00C18 P.AFZ:    .BYTE   3, 0
                        000F   00C1A           .WORD   15
00  00  00  00  00  00  32  32 00C1C           .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                        3E  01 00C24           .ASCII  <1>\>\
                        00  03 00C26 P.AGA:    .BYTE   3, 0
                        0011   00C28           .WORD   17
00  00  00  00  00  00  32  32 00C2A           .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                    3D  3E  02 00C32           .ASCII  <2>\>=\
                        00  03 00C35 P.AGB:    .BYTE   3, 0
                        0013   00C37           .WORD   19
00  00  00  00  00  00  32  32 00C39           .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                        3C  01 00C41           .ASCII  <1>\<\
                        00  03 00C43 P.AGC:    .BYTE   3, 0
                        0015   00C45           .WORD   21
00  00  00  00  00  00  32  32 00C47           .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                    3D  3C  02 00C4F           .ASCII  <2>\<=\
                        02  02 00C52 P.AGD:    .BYTE   2, 2
                        000B   00C54           .WORD   11
00  00  00  00  00  00  C8  05 00C56           .BYTE   5, -56, 0, 0, 0, 0, 0, 0
                        28  01 00C5E           .ASCII  <1>\(\
                        02  04 00C60 P.AGE:    .BYTE   4, 2
                        000C   00C62           .WORD   12
00  00  00  00  00  00  06  C8 00C64           .BYTE   -56, 6, 0, 0, 0, 0, 0, 0
                        29  01 00C6C           .ASCII  <1>\)\
                               00C6E           .BLKB   2
                    00000018   00C70           .LONG   24
```

```
00000ADB  00000ACD  00000ABF  00000AB1  00000AA3  00000A95  00C74 P.AFG:  .LONG   2709, 2723, 2737, 2751, 2765, 2779, 2793, -
00000B30  00000B22  00000B14  00000B06  00000AF8  00000AE9  00C8C                2808, 2822, 2836, 2850, 2864, 2878, 2892, -
00000B86  00000B77  00000B69  00000B5A  00000B4C  00000B3E  00CA4                2906, 2921, 2935, 2950, 2965, 2979, 2994, -
00000BDD  00000BCF  00000BC0  00000BB2  00000BA3  00000B95  00CBC                3008, 3023, 3037
                                                      01  00  00CD4 P.AGG:  .BYTE   0, 1
                                                    0002  00CD6           .WORD   2
                                              00    00  00  00CD8           .BYTE   0, 0, 0, 0
                                                      5D  01  00CDC           .ASCII  <1>\]\
                                                      01  00  00CDE P.AGH:  .BYTE   0, 1
                                                    0002  00CE0           .WORD   2
                                              00    00  00  00CE2           .BYTE   0, 0, 0, 0
                                                      29  01  00CE6           .ASCII  <1>\)\
                                                      02  00  00CE8 P.AGI:  .BYTE   0, 2
                                                    0003  00CEA           .WORD   3
                                              00    00  00  00CEC           .BYTE   0, 0, 0, 0
                                                      3A  01  00CF0           .ASCII  <1>\:\
                                                      00  00  00CF2 P.AGJ:  .BYTE   0, 0
                                                    0001  00CF4           .WORD   1
                                              00    00  00  00CF6           .BYTE   0, 0, 0, 0
                                                      2C  01  00CFA           .ASCII  <1>\,\
                                                00000004  00CFC           .LONG   4
                          00000C6F  00000C65  00000C5B  00000C51  00D00 P.AGF:  .LONG   3153, 3163, 3173, 3183
                                                      01  00D10 P.AGK:  .BYTE   1
                                                      02  00D11           .BYTE   2
                                                    0005  00D12           .WORD   5
                                                      03  00D14           .BYTE   3
                                                      03  00D15           .BYTE   3
                                                    0003  00D16           .WORD   3
                                                      00  00D18           .BYTE   0
                                                      0A  00D19           .BYTE   10
                                                    0021  00D1A           .WORD   33
                                                      01  00D1C           .BYTE   1
                                                      04  00D1D           .BYTE   4
                                                    0012  00D1E           .WORD   18
                                                      00  00D20           .BYTE   0
                                                      0A  00D21           .BYTE   10
                                                    0021  00D22           .WORD   33
                                                      01  00D24           .BYTE   1
                                                      02  00D25           .BYTE   2
                                                    0005  00D26           .WORD   5
                                                      03  00D28           .BYTE   3
                                                      03  00D29           .BYTE   3
                                                    0012  00D2A           .WORD   18
                                                      02  00D2C           .BYTE   2
                                                      01  00D2D           .BYTE   1
                                                    000C  00D2E           .WORD   12
                                                      06  00D30           .BYTE   6
                                                      01  00D31           .BYTE   1
                                                    000C  00D32           .WORD   12
                                                      07  00D34           .BYTE   7
                                                      01  00D35           .BYTE   1
                                                    000C  00D36           .WORD   12
                                                      08  00D38           .BYTE   8
                                                      01  00D39           .BYTE   1
                                                    000C  00D3A           .WORD   12
                                                      00  00D3C           .BYTE   0
                                                      09  00D3D           .BYTE   9
```

```
0021  00D3E          .WORD   33
  01  00D40          .BYTE   1
  01  00D41          .BYTE   1
000C  00D42          .WORD   12
  02  00D44          .BYTE   2
  01  00D45          .BYTE   1
000C  00D46          .WORD   12
  06  00D48          .BYTE   6
  01  00D49          .BYTE   1
000C  00D4A          .WORD   12
  07  00D4C          .BYTE   7
  01  00D4D          .BYTE   1
000C  00D4E          .WORD   12
  08  00D50          .BYTE   8
  01  00D51          .BYTE   1
000C  00D52          .WORD   12
  00  00D54          .BYTE   0
  09  00D55          .BYTE   9
0021  00D56          .WORD   33
  01  00D58          .BYTE   1
  04  00D59          .BYTE   4
0012  00D5A          .WORD   18
  03  00D5C          .BYTE   3
  08  00D5D          .BYTE   8
0021  00D5E          .WORD   33
  08  00D60          .BYTE   8
  05  00D61          .BYTE   5
0019  00D62          .WORD   25
  07  00D64          .BYTE   7
  06  00D65          .BYTE   6
0019  00D66          .WORD   25
  0D  00D68          .BYTE   13
  12  00D69          .BYTE   18
0019  00D6A          .WORD   25
  09  00D6C          .BYTE   9
  07  00D6D          .BYTE   7
0019  00D6E          .WORD   25
  00  00D70          .BYTE   0
  09  00D71          .BYTE   9
0021  00D72          .WORD   33
  01  00D74          .BYTE   1
  01  00D75          .BYTE   1
001F  00D76          .WORD   31
  04  00D78          .BYTE   4
  01  00D79          .BYTE   1
001D  00D7A          .WORD   29
  05  00D7C          .BYTE   5
  01  00D7D          .BYTE   1
001D  00D7E          .WORD   29
  00  00D80          .BYTE   0
  08  00D81          .BYTE   8
0021  00D82          .WORD   33
  01  00D84          .BYTE   1
  01  00D85          .BYTE   1
001F  00D86          .WORD   31
  00  00D88          .BYTE   0
  09  00D89          .BYTE   9
```

```
                00D8A           .WORD   33
                00D8C           .BYTE   1
                00D8D           .BYTE   1
                00D8E           .WORD   31
                00D90           .BYTE   0
                00D91           .BYTE   9
                00D92           .WORD   33
                00D94           .BYTE   0
                00D95           .BYTE   11
                00D96           .WORD   33
00000000        00D98           .LONG   0
                00D9C P.AGL:    .BLKB   0
00000000        00D9C           .LONG   0
                00DA0 P.AGM:    .BLKB   0
                00DA0 P.AGN:    .BYTE   2
                00DA1           .BYTE   1
                00DA2           .WORD   8
                00DA4           .BYTE   3
                00DA5           .BYTE   3
                00DA6           .WORD   10
                00DA8           .BYTE   9
                00DA9           .BYTE   15
                00DAA           .WORD   10
                00DAC           .BYTE   5
                00DAD           .BYTE   4
                00DAE           .WORD   17
                00DB0           .BYTE   4
                00DB1           .BYTE   7
                00DB2           .WORD   22
                00DB4           .BYTE   7
                00DB5           .BYTE   10
                00DB6           .WORD   27
                00DB8           .BYTE   1
                00DB9           .BYTE   12
                00DBA           .WORD   32
00000000        00DBC           .LONG   0
                00DC0           .BYTE   1
                00DC1           .BYTE   2
                00DC2           .WORD   32
00000000        00DC4           .LONG   0
                00DC8           .BYTE   3
                00DC9           .BYTE   14
                00DCA           .WORD   10
                00DCC           .BYTE   9
                00DCD           .BYTE   15
                00DCE           .WORD   10
                00DD0           .BYTE   5
                00DD1           .BYTE   16
                00DD2           .WORD   17
                00DD4           .BYTE   4
                00DD5           .BYTE   18
                00DD6           .WORD   22
                00DD8           .BYTE   7
                00DD9           .BYTE   21
                00DDA           .WORD   27
                00DDC           .BYTE   1
                00DDD           .BYTE   23
```

```
                                                    0020  00DDE           .WORD   32
                                                00000000  00DE0           .LONG   0
                                                      05  00DE4           .BYTE   5
                                                      19  00DE5           .BYTE   25
                                                    0011  00DE6           .WORD   17
                                                      04  00DE8           .BYTE   4
                                                      1C  00DE9           .BYTE   28
                                                    0016  00DEA           .WORD   22
                                                      07  00DEC           .BYTE   7
                                                      1F  00DED           .BYTE   31
                                                    001B  00DEE           .WORD   27
                                                      01  00DF0           .BYTE   1
                                                      21  00DF1           .BYTE   33
                                                    0020  00DF2           .WORD   32
                                                00000000  00DF4           .LONG   0
                                                      05  00DF8           .BYTE   5
                                                      24  00DF9           .BYTE   36
                                                    0011  00DFA           .WORD   17
                                                      04  00DFC           .BYTE   4
                                                      26  00DFD           .BYTE   38
                                                    0016  00DFE           .WORD   22
                                                      07  00E00           .BYTE   7
                                                      28  00E01           .BYTE   40
                                                    001B  00E02           .WORD   27
                                                      01  00E04           .BYTE   1
                                                      2A  00E05           .BYTE   42
                                                    0020  00E06           .WORD   32
                                                00000000  00E08           .LONG   0
                                                      05  00E0C           .BYTE   5
                                                      2C  00E0D           .BYTE   44
                                                    0011  00E0E           .WORD   17
                                                      04  00E10           .BYTE   4
                                                      2D  00E11           .BYTE   45
                                                    0016  00E12           .WORD   22
                                                      07  00E14           .BYTE   7
                                                      2E  00E15           .BYTE   46
                                                    001B  00E16           .WORD   27
                                                      01  00E18           .BYTE   1
                                                      2F  00E19           .BYTE   47
                                                    0020  00E1A           .WORD   32
                                                00000000  00E1C           .LONG   0
                                                00000000  00E20           .LONG   0
00000C7D  00000D1D  00000C8D  00000BF1  00000A69  00000995  00E24 P.AGO:   .LONG   2453, 2665, 3057, 3213, 3357, 3197, 3353, -
00000000  00000000  00000001  00000000  00000D1D  00000D19  00E3C           3357, 0, 1, 0, 0, 0
                                                00000000  00E54
                                                00000009  00E58           .LONG   9
3C00A800  2F00A800  26000800  230000C0  5F0000B2  27000000  00E5C P.AGP:   .LONG   654311424, 1593835698, 587202752, -
                              2E058830  3E042800  3D042800  00E74           637536256, 788572160, 1006675968, -
                                                                            1023682560, 1040459776, 772114480
                                                      00  02  00E80 P.AGR:  .BYTE   2, 0
                                                        0017  00E82         .WORD   23
                                    00  00  00  00  00  00  C8  3C  00E84         .BYTE   60, -56, 0, 0, 0, 0, 0, 0
                                                    54  4F  4E  03  00E8C         .ASCII  <3>\NOT\
                                                      00  02  00E90 P.AGS:  .BYTE   2, 0
                                                        002B  00E92         .WORD   43
                                    00  00  00  00  00  00  C8  3C  00E94         .BYTE   60, -56, 0, 0, 0, 0, 0, 0
                                                    53  42  41  03  00E9C         .ASCII  <3>\ABS\
```

```
                                                    00  03  00EA0 P.AGT:   .BYTE   3, 0
                                                      0024  00EA2         .WORD   36
                                00  00        00  00  32  32  00EA4         .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                              44  4F  4D  03  00EAC         .ASCII  <3>\MOD\
                                                    00  03  00EB0 P.AGU:   .BYTE   3, 0
                                                      0025  00EB2         .WORD   37
                                00  00  00  00  00  00  32  32  00EB4         .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                              4D  45  52  03  00EBC         .ASCII  <3>\REM\
                                                    00  03  00EC0 P.AGV:   .BYTE   3, 0
                                                      0018  00EC2         .WORD   24
                                00  00  00  00  00  00  0A  0A  00EC4         .BYTE   10, 10, 0, 0, 0, 0, 0, 0
                                              44  4E  41  03  00ECC         .ASCII  <3>\AND\
                                                    00  03  00ED0 P.AGW:   .BYTE   3, 0
                                                      0019  00ED2         .WORD   25
                                00  00  00  00  00  00  0A  0A  00ED4         .BYTE   10, 10, 0, 0, 0, 0, 0, 0
                                                  52  4F  02  00EDC         .ASCII  <2>\OR\
                                                    00  03  00EDF P.AGX:   .BYTE   3, 0
                                                      001A  00EE1         .WORD   26
                                00  00  00  00  00  00  0A  0A  00EE3         .BYTE   10, 10, 0, 0, 0, 0, 0, 0
                                              52  4F  58  03  00EEB         .ASCII  <3>\XOR\
                                                              00EEF         .BLKB   1
                                                  00000007    00EF0         .LONG   7
00000E4D  00000E3D  00000E2D  00000E1D  00000E0D  00000DFD    00EF4 P.AGQ:   .LONG   3581, 3597, 3613, 3629, 3645, 3661, 3676
                                                  00000E5C    00F0C
                                                    01  02  00F10 P.AGZ:   .BYTE   2, 1
                                                      0002  00F12         .WORD   2
                                00  00  00  00  00  00  00  00  00F14         .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                    5C  01  00F1C         .ASCII  <1><92>
                                                    01  03  00F1E P.AHA:   .BYTE   3, 1
                                                      0003  00F20         .WORD   3
                                00  00  00  00  00  00  00  00  00F22         .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                    5C  01  00F2A         .ASCII  <1><92>
                                                    01  03  00F2C P.AHB:   .BYTE   3, 1
                                                      0005  00F2E         .WORD   5
                                00  00  00  00  00  00  00  00  00F30         .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                    2E  01  00F38         .ASCII  <1>\.\
                                                    01  04  00F3A P.AHC:   .BYTE   4, 1
                                                      0004  00F3C         .WORD   4
                                00  00  00  00  00  00  00  00  00F3E         .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                    28  01  00F46         .ASCII  <1>\(\
                                                    02  02  00F48 P.AHD:   .BYTE   2, 2
                                                      000B  00F4A         .WORD   11
                                00  00  00  00  00  00  C8  05  00F4C         .BYTE   5, -56, 0, 0, 0, 0, 0, 0
                                                    28  01  00F54         .ASCII  <1>\(\
                                                    02  04  00F56 P.AHE:   .BYTE   4, 2
                                                      000C  00F58         .WORD   12
                                00  00  00  00  00  00  06  C8  00F5A         .BYTE   -56, 6, 0, 0, 0, 0, 0, 0
                                                    29  01  00F62         .ASCII  <1>\)\
                                                    00  03  00F64 P.AHF:   .BYTE   3, 0
                                                      000A  00F66         .WORD   10
                                00  00  00  00  00  00  3C  3C  00F68         .BYTE   60, 60, 0, 0, 0, 0, 0, 0
                                                  2A  2A  02  00F70         .ASCII  <2>\**\
                                                    00  03  00F73 P.AHG:   .BYTE   3, 0
                                                      0008  00F75         .WORD   8
                                00  00  00  00  00  00  32  32  00F77         .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                                    2A  01  00F7F         .ASCII  <1>\*\
                                                    00  03  00F81 P.AHH:   .BYTE   3, 0
```

```
                                                    0009   00F83          .WORD   9
                        00  00  00  00  00  00   32   32   00F85          .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                                 2F   01   00F8D          .ASCII  <1>\/\
                                                 00   02   00F8F P.AHI:   .BYTE   2, 0
                                                    0004   00F91          .WORD   4
                        00  00  00  00  00  00   C8   28   00F93          .BYTE   40, -56, 0, 0, 0, 0, 0, 0
                                                 2B   01   00F9B          .ASCII  <1>\+\
                                                 00   02   00F9D P.AHJ:   .BYTE   2, 0
                                                    0005   00F9F          .WORD   5
                        00  00  00  00  00  00   C8   28   00FA1          .BYTE   40, -56, 0, 0, 0, 0, 0, 0
                                                 2D   01   00FA9          .ASCII  <1>\-\
                                                 00   03   00FAB P.AHK:   .BYTE   3, 0
                                                    0006   00FAD          .WORD   6
                        00  00  00  00  00  00   1E   1E   00FAF          .BYTE   30, 30, 0, 0, 0, 0, 0, 0
                                                 2B   01   00FB7          .ASCII  <1>\+\
                                                 00   03   00FB9 P.AHL:   .BYTE   3, 0
                                                    0007   00FBB          .WORD   7
                        00  00  00  00  00  00   1E   1E   00FBD          .BYTE   30, 30, 0, 0, 0, 0, 0, 0
                                                 2D   01   00FC5          .ASCII  <1>\-\
                                                 00   03   00FC7 P.AHM:   .BYTE   3, 0
                                                    0023   00FC9          .WORD   35
                        00  00  00  00  00  00   1E   1E   00FCB          .BYTE   30, 30, 0, 0, 0, 0, 0, 0
                                                 26   01   00FD3          .ASCII  <1>\&\
                                                 00   03   00FD5 P.AHN:   .BYTE   3, 0
                                                    000D   00FD7          .WORD   13
                        00  00  00  00  00  00   14   14   00FD9          .BYTE   20, 20, 0, 0, 0, 0, 0, 0
                                                 3D   01   00FE1          .ASCII  <1>\=\
                                                 00   03   00FE3 P.AHO:   .BYTE   3, 0
                                                    000E   00FE5          .WORD   14
                        00  00  00  00  00  00   14   14   00FE7          .BYTE   20, 20, 0, 0, 0, 0, 0, 0
                                             3D  2F   02   00FEF          .ASCII  <2>\/=\
                                                 00   03   00FF2 P.AHP:   .BYTE   3, 0
                                                    0013   00FF4          .WORD   19
                        00  00  00  00  00  00   14   14   00FF6          .BYTE   20, 20, 0, 0, 0, 0, 0, 0
                                                 3C   01   00FFE          .ASCII  <1>\<\
                                                 00   03   01000 P.AHQ:   .BYTE   3, 0
                                                    0015   01002          .WORD   21
                        00  00  00  00  00  00   14   14   01004          .BYTE   20, 20, 0, 0, 0, 0, 0, 0
                                             3D  3C   02   0100C          .ASCII  <2>\<=\
                                                 00   03   0100F P.AHR:   .BYTE   3, 0
                                                    000F   01011          .WORD   15
                        00  00  00  00  00  00   14   14   01013          .BYTE   20, 20, 0, 0, 0, 0, 0, 0
                                                 3E   01   0101B          .ASCII  <1>\>\
                                                 00   03   0101D P.AHS:   .BYTE   3, 0
                                                    0011   0101F          .WORD   17
                        00  00  00  00  00  00   14   14   01021          .BYTE   20, 20, 0, 0, 0, 0, 0, 0
                                             3D  3E   02   01029          .ASCII  <2>\>=\
                                                00000014   0102C          .LONG   20
00000ED3  00000EC5  00000EB7  00000EA9  00000E9B  00000E8D   01030 P.AGY:   .LONG   3725, 3739, 3753, 3767, 3781, 3795, 3809, -
00000F28  00000F1A  00000F0C  00000EFE  00000EF0  00000EE1   01048                  3824, 3838, 3852, 3866, 3880, 3894, 3908, -
00000F7D  00000F6F  00000F60  00000F52  00000F44  00000F36   01060                  3922, 3936, 3951, 3965, 3980, 3994
                                        00000F9A  00000F8C   01078
                                                 01   04   01080 P.AHT:   .BYTE   4, 1
                                                    000B   01082          .WORD   11
                        00  00  00  00  00  00   00   00   01084          .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                 27   01   0108C          .ASCII  <1>\'\
                                                           0108E          .BLKB   2
```

```
 44  45  4E  49  41  52  54  53  4E  4F  43  0B  01090 P.AHU:   .ASCII  <11>\CONSTRAINED\
                 00  00  54  53  52  49  46  05  0109C P.AHV:   .ASCII  <5>\FIRST\<0><0>
                 00  00  00  54  53  41  4C  04  010A4 P.AHW:   .ASCII  <4>\LAST\<0><0><0>
                 00  48  54  47  4E  45  4C  06  010AC P.AHX:   .ASCII  <6>\LENGTH\<0>
                                 53  4F  50  03  010B4 P.AHY:   .ASCII  <3>\POS\
                 00  00  00  44  45  52  50  04  010B8 P.AHZ:   .ASCII  <4>\PRED\<0><0><0>
                 00  00  00  45  5A  49  53  04  010C0 P.AIA:   .ASCII  <4>\SIZE\<0><0><0>
                 00  00  00  43  43  55  53  04  010C8 P.AIB:   .ASCII  <4>\SUCC\<0><0><0>
                                 4C  41  56  03  010D0 P.AIC:   .ASCII  <3>\VAL\
                                            00#  010D4 ADA_TICK_TABLE:
                                                                .BYTE   0[4]
00001035  00001031  00001029  00001021  00001019  0000100D  010D8   .LONG   4109, 4121, 4129, 4137, 4145, 4149, 4157, -
                              0000104D  00001045  0000103D  010F0           4165, 4173
                                                   00   00  010FC P.AIE:   .BYTE   0, 0
                                                     0002  010FE          .WORD   2
                                           00  00  00   00  01100          .BYTE   0, 0, 0, 0
                                                   29   01  01104          .ASCII  <1>\)\
                                                   02   00  01106 P.AIF:   .BYTE   0, 2
                                                     0003  01108          .WORD   3
                                           00  00  00   00  0110A          .BYTE   0, 0, 0, 0
                                                   3A   01  0110E          .ASCII  <1>\:\
                                                   00   00  01110 P.AIG:   .BYTE   0, 0
                                                     0003  01112          .WORD   3
                                           00  00  00   00  01114          .BYTE   0, 0, 0, 0
                                               2E  2E   02  01118          .ASCII  <2>\..\
                                                   00   00  0111B P.AIH:   .BYTE   0, 0
                                                     0001  0111D          .WORD   1
                                           00  00  00   00  0111F          .BYTE   0, 0, 0, 0
                                                   2C   01  01123          .ASCII  <1>\,\
                                                            01125          .BLKB   3
                                                 00000004  01128          .LONG   4
                    00001098  0000108D  00001083  00001079  0112C P.AID:   .LONG   4217, 4227, 4237, 4248
                                                 00000000  0113C          .LONG   0
                                                            01140 P.AII:   .BLKB   0
                                                 00000000  01140          .LONG   0
                                                            01144 P.AIJ:   .BLKB   0
                                                       01  01144 P.AIK:   .BYTE   1
                                                       01  01145          .BYTE   1
                                                     0002  01146          .WORD   2
                                                       00  01148          .BYTE   0
                                                       0A  01149          .BYTE   10
                                                     003E  0114A          .WORD   62
                                                       01  0114C          .BYTE   1
                                                       01  0114D          .BYTE   1
                                                     0002  0114E          .WORD   2
                                                       0B  01150          .BYTE   11
                                                       01  01151          .BYTE   1
                                                     0008  01152          .WORD   8
                                                       0C  01154          .BYTE   12
                                                       11  01155          .BYTE   17
                                                     001F  01156          .WORD   31
                                                       03  01158          .BYTE   3
                                                       03  01159          .BYTE   3
                                                     000A  0115A          .WORD   10
                                                       08  0115C          .BYTE   8
                                                       01  0115D          .BYTE   1
                                                     0014  0115E          .WORD   20
```

```
00  01160          .BYTE   0
09  01161          .BYTE   9
003E  01162        .WORD   62
01  01164          .BYTE   1
01  01165          .BYTE   1
0002  01166        .WORD   2
00  01168          .BYTE   0
0A  01169          .BYTE   10
003E  0116A        .WORD   62
01  0116C          .BYTE   1
01  0116D          .BYTE   1
000A  0116E        .WORD   10
0B  01170          .BYTE   11
01  01171          .BYTE   1
0012  01172        .WORD   18
03  01174          .BYTE   3
0A  01175          .BYTE   10
003E  01176        .WORD   62
08  01178          .BYTE   8
05  01179          .BYTE   5
0014  0117A        .WORD   20
07  0117C          .BYTE   7
06  0117D          .BYTE   6
0014  0117E        .WORD   20
0D  01180          .BYTE   13
12  01181          .BYTE   18
0014  01182        .WORD   20
09  01184          .BYTE   9
07  01185          .BYTE   7
0014  01186        .WORD   20
00  01188          .BYTE   0
09  01189          .BYTE   9
003E  0118A        .WORD   62
01  0118C          .BYTE   1
01  0118D          .BYTE   1
000A  0118E        .WORD   10
00  01190          .BYTE   0
0A  01191          .BYTE   10
003E  01192        .WORD   62
01  01194          .BYTE   1
01  01195          .BYTE   1
001A  01196        .WORD   26
04  01198          .BYTE   4
01  01199          .BYTE   1
0018  0119A        .WORD   24
05  0119C          .BYTE   5
01  0119D          .BYTE   1
0018  0119E        .WORD   24
00  011A0          .BYTE   0
0A  011A1          .BYTE   10
003E  011A2        .WORD   62
01  011A4          .BYTE   1
01  011A5          .BYTE   1
001A  011A6        .WORD   26
00  011A8          .BYTE   0
0A  011A9          .BYTE   10
003E  011AA        .WORD   62
```

```
  01  011AC          .BYTE   1
  01  011AD          .BYTE   1
001A  011AE          .WORD   26
  0B  011B0          .BYTE   11
  01  011B1          .BYTE   1
001D  011B2          .WORD   29
  00  011B4          .BYTE   0
  09  011B5          .BYTE   9
003E  011B6          .WORD   62
  01  011B8          .BYTE   1
  01  011B9          .BYTE   1
001A  011BA          .WORD   26
  00  011BC          .BYTE   0
  0A  011BD          .BYTE   10
003E  011BE          .WORD   62
  01  011C0          .BYTE   1
  01  011C1          .BYTE   1
0021  011C2          .WORD   33
  00  011C4          .BYTE   0
  0A  011C5          .BYTE   10
003E  011C6          .WORD   62
  01  011C8          .BYTE   1
  01  011C9          .BYTE   1
0021  011CA          .WORD   33
  02  011CC          .BYTE   2
  01  011CD          .BYTE   1
0021  011CE          .WORD   33
  06  011D0          .BYTE   6
  01  011D1          .BYTE   1
0021  011D2          .WORD   33
  07  011D4          .BYTE   7
  01  011D5          .BYTE   1
0021  011D6          .WORD   33
  08  011D8          .BYTE   8
  01  011D9          .BYTE   1
0021  011DA          .WORD   33
  0B  011DC          .BYTE   11
  01  011DD          .BYTE   1
002A  011DE          .WORD   42
  03  011E0          .BYTE   3
  03  011E1          .BYTE   3
0030  011E2          .WORD   48
  0C  011E4          .BYTE   12
  01  011E5          .BYTE   1
0014  011E6          .WORD   20
  00  011E8          .BYTE   0
  09  011E9          .BYTE   9
003E  011EA          .WORD   62
  01  011EC          .BYTE   1
  01  011ED          .BYTE   1
0021  011EE          .WORD   33
  02  011F0          .BYTE   2
  01  011F1          .BYTE   1
0021  011F2          .WORD   33
  06  011F4          .BYTE   6
  01  011F5          .BYTE   1
0021  011F6          .WORD   33
```

```
  07  011F8          .BYTE   7
  01  011F9          .BYTE   1
0021  011FA          .WORD   33
  08  011FC          .BYTE   8
  01  011FD          .BYTE   1
0021  011FE          .WORD   33
  00  01200          .BYTE   0
  0A  01201          .BYTE   10
003E  01202          .WORD   62
  01  01204          .BYTE   1
  01  01205          .BYTE   1
0030  01206          .WORD   48
  02  01208          .BYTE   2
  01  01209          .BYTE   1
0030  0120A          .WORD   48
  06  0120C          .BYTE   6
  01  0120D          .BYTE   1
0030  0120E          .WORD   48
  07  01210          .BYTE   7
  01  01211          .BYTE   1
0030  01212          .WORD   48
  08  01214          .BYTE   8
  01  01215          .BYTE   1
0030  01216          .WORD   48
  0B  01218          .BYTE   11
  01  01219          .BYTE   1
0038  0121A          .WORD   56
  0C  0121C          .BYTE   12
  05  0121D          .BYTE   5
0014  0121E          .WORD   20
  00  01220          .BYTE   0
  0A  01221          .BYTE   10
003E  01222          .WORD   62
  01  01224          .BYTE   1
  01  01225          .BYTE   1
0030  01226          .WORD   48
  02  01228          .BYTE   2
  01  01229          .BYTE   1
0030  0122A          .WORD   48
  06  0122C          .BYTE   6
  01  0122D          .BYTE   1
0030  0122E          .WORD   48
  07  01230          .BYTE   7
  01  01231          .BYTE   1
0030  01232          .WORD   48
  08  01234          .BYTE   8
  01  01235          .BYTE   1
0030  01236          .WORD   48
  00  01238          .BYTE   0
  0A  01239          .BYTE   10
003E  0123A          .WORD   62
  00  0123C          .BYTE   0
  0B  0123D          .BYTE   11
003E  0123E          .WORD   62
  02  01240 P.AIL:   .BYTE   2
  01  01241          .BYTE   1
0008  01242          .WORD   8
```

```
                03  01244          .BYTE   3
                03  01245          .BYTE   3
              000A  01246          .WORD   10
                09  01248          .BYTE   9
                0F  01249          .BYTE   15
              000A  0124A          .WORD   10
                05  0124C          .BYTE   5
                04  0124D          .BYTE   4
              0011  0124E          .WORD   17
                04  01250          .BYTE   4
                07  01251          .BYTE   7
              0016  01252          .WORD   22
                0B  01254          .BYTE   11
                30  01255          .BYTE   48
              001A  01256          .WORD   26
                01  01258          .BYTE   1
                0C  01259          .BYTE   12
              001A  0125A          .WORD   26
          00000000  0125C          .LONG   0
                01  01260          .BYTE   1
                02  01261          .BYTE   2
              001A  01262          .WORD   26
          00000000  01264          .LONG   0
                03  01268          .BYTE   3
                0E  01269          .BYTE   14
              000A  0126A          .WORD   10
                09  0126C          .BYTE   9
                0F  0126D          .BYTE   15
              000A  0126E          .WORD   10
                05  01270          .BYTE   5
                10  01271          .BYTE   16
              0011  01272          .WORD   17
                04  01274          .BYTE   4
                12  01275          .BYTE   18
              0016  01276          .WORD   22
                0B  01278          .BYTE   11
                31  01279          .BYTE   49
              001A  0127A          .WORD   26
                01  0127C          .BYTE   1
                17  0127D          .BYTE   23
              001A  0127E          .WORD   26
          00000000  01280          .LONG   0
                05  01284          .BYTE   5
                19  01285          .BYTE   25
              0011  01286          .WORD   17
                04  01288          .BYTE   4
                1C  01289          .BYTE   28
              0016  0128A          .WORD   22
                0B  0128C          .BYTE   11
                32  0128D          .BYTE   50
              001A  0128E          .WORD   26
                01  01290          .BYTE   1
                21  01291          .BYTE   33
              001A  01292          .WORD   26
          00000000  01294          .LONG   0
                05  01298          .BYTE   5
                24  01299          .BYTE   36
```

```
                                                    0011  0129A          .WORD   17
                                                      04  0129C          .BYTE   4
                                                      26  0129D          .BYTE   38
                                                    0016  0129E          .WORD   22
                                                      01  012A0          .BYTE   1
                                                      2A  012A1          .BYTE   42
                                                    001A  012A2          .WORD   26
                                               00000000  012A4          .LONG   0
                                               00000000  012A8          .LONG   0
000010A9  000011BD  000010C1  00000FAD  00000E71  00000DD9  012AC P.AIM:  .LONG   3545, 3697, 4013, 4289, 4541, 4265, 4285, -
00000000  00000000  00000001  00000000  000010C1  000010BD  012C4                 4289, 0, 1, 0, 0, 0
                                               00000000  012DC
                                               00000007  012E0          .LONG   7
3E042800  3C04A800  5E010800  3A042800  25000004  2E01803E  012E4 P.AIN:  .LONG   771850302, 620756996, 973350912, -
                                               3D042800  012FC                 1577125888, 1006938112, 1040459776, -
                                                                               1023682560
                                                 00  02  01300 P.AIP:  .BYTE   2, 0
                                                   001E  01302          .WORD   30
                                  00  00  00  00  00  00  C8  2D  01304          .BYTE   45, -56, 0, 0, 0, 0, 0, 0
                                                  54  4F  4E  03  0130C          .ASCII  <3>\NOT\
                                                 00  03  01310 P.AIQ:  .BYTE   3, 0
                                                   001F  01312          .WORD   31
                                  00  00  00  00  00  00  28  28  01314          .BYTE   40, 40, 0, 0, 0, 0, 0, 0
                                                  44  4E  41  03  0131C          .ASCII  <3>\AND\
                                                 00  03  01320 P.AIR:  .BYTE   3, 0
                                                   0020  01322          .WORD   32
                                  00  00  00  00  00  00  1E  1E  01324          .BYTE   30, 30, 0, 0, 0, 0, 0, 0
                                                  52  4F  02  0132C          .ASCII  <2>\OR\
                                                 00  03  0132F P.AIS:  .BYTE   3, 0
                                                   0021  01331          .WORD   33
                                  00  00  00  00  00  00  1E  1E  01333          .BYTE   30, 30, 0, 0, 0, 0, 0, 0
                                                  52  4F  58  03  0133B          .ASCII  <3>\XOR\
                                                 00  03  0133F P.AIT:  .BYTE   3, 0
                                                   003A  01341          .WORD   58
                                  00  00  00  00  00  00  14  14  01343          .BYTE   20, 20, 0, 0, 0, 0, 0, 0
                                                  50  4D  49  03  0134B          .ASCII  <3>\IMP\
                                                 00  03  0134F P.AIU:  .BYTE   3, 0
                                                   0022  01351          .WORD   34
                                  00  00  00  00  00  00  0A  0A  01353          .BYTE   10, 10, 0, 0, 0, 0, 0, 0
                                                  56  51  45  03  0135B          .ASCII  <3>\EQV\
                                                              0135F          .BLKB   1
                                               00000006  01360          .LONG   6
000012CC  000012BC  000012AC  0000129D  0000128D  0000127D  01364 P.AIO:  .LONG   4733, 4749, 4765, 4780, 4796, 4812
                                                 01  02  0137C P.AIW:  .BYTE   2, 1
                                                   0002  0137E          .WORD   2
                                  00  00  00  00  00  00  00  00  01380          .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                 5C  01  01388          .ASCII  <1><92>
                                                 01  03  0138A P.AIX:  .BYTE   3, 1
                                                   0003  0138C          .WORD   3
                                  00  00  00  00  00  00  00  00  0138E          .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                 5C  01  01396          .ASCII  <1><92>
                                                 01  03  01398 P.AIY:  .BYTE   3, 1
                                                   0005  0139A          .WORD   5
                                  00  00  00  00  00  00  00  00  0139C          .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                  3A  3A  02  013A4          .ASCII  <2>\::\
                                                 01  04  013A7 P.AIZ:  .BYTE   4, 1
                                                   0004  013A9          .WORD   4
```

```
                00 00 00 00 00 00  00 00  013AB            .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                   28 01  013B3            .ASCII  <1>\(\
                                   02 02  013B5 P.AJA:     .BYTE   2, 2
                                     000B 013B7            .WORD   11
                00 00 00 00 00 00  C8 05  013B9            .BYTE   5, -56, 0, 0, 0, 0, 0, 0
                                   28 01  013C1            .ASCII  <1>\(\
                                   02 04  013C3 P.AJB:     .BYTE   4, 2
                                     000C 013C5            .WORD   12
                00 00 00 00 00 00  06 C8  013C7            .BYTE   -56, 6, 0, 0, 0, 0, 0, 0
                                   29 01  013CF            .ASCII  <1>\)\
                                   00 02  013D1 P.AJC:     .BYTE   2, 0
                                     0004 013D3            .WORD   4
                00 00 00 00 00 00  C8 46  013D5            .BYTE   70, -56, 0, 0, 0, 0, 0, 0
                                   2B 01  013DD            .ASCII  <1>\+\
                                   00 02  013DF P.AJD:     .BYTE   2, 0
                                     0005 013E1            .WORD   5
                00 00 00 00 00 00  C8 46  013E3            .BYTE   70, -56, 0, 0, 0, 0, 0, 0
                                   2D 01  013EB            .ASCII  <1>\-\
                                   00 03  013ED P.AJE:     .BYTE   3, 0
                                     000A 013EF            .WORD   10
                00 00 00 00 00 00  5C 5A  013F1            .BYTE   90, 92, 0, 0, 0, 0, 0, 0
                                2A 2A 02  013F9            .ASCII  <2>\**\
                                   00 03  013FC P.AJF:     .BYTE   3, 0
                                     000A 013FE            .WORD   10
                00 00 00 00 00 00  5C 5A  01400            .BYTE   90, 92, 0, 0, 0, 0, 0, 0
                                   5E 01  01408            .ASCII  <1>\^\
                                   00 03  0140A P.AJG:     .BYTE   3, 0
                                     0008 0140C            .WORD   8
                00 00 00 00 00 00  50 50  0140E            .BYTE   80, 80, 0, 0, 0, 0, 0, 0
                                   2A 01  01416            .ASCII  <1>\*\
                                   00 03  01418 P.AJH:     .BYTE   3, 0
                                     0009 0141A            .WORD   9
                00 00 00 00 00 00  50 50  0141C            .BYTE   80, 80, 0, 0, 0, 0, 0, 0
                                   2F 01  01424            .ASCII  <1>\/\
                                   00 03  01426 P.AJI:     .BYTE   3, 0
                                     0006 01428            .WORD   6
                00 00 00 00 00 00  3C 3C  0142A            .BYTE   60, 60, 0, 0, 0, 0, 0, 0
                                   2B 01  01432            .ASCII  <1>\+\
                                   00 03  01434 P.AJJ:     .BYTE   3, 0
                                     0007 01436            .WORD   7
                00 00 00 00 00 00  3C 3C  01438            .BYTE   60, 60, 0, 0, 0, 0, 0, 0
                                   2D 01  01440            .ASCII  <1>\-\
                                   00 03  01442 P.AJK:     .BYTE   3, 0
                                     0013 01444            .WORD   19
                00 00 00 00 00 00  32 32  01446            .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                   3C 01  0144E            .ASCII  <1>\<\
                                   00 03  01450 P.AJL:     .BYTE   3, 0
                                     0015 01452            .WORD   21
                00 00 00 00 00 00  32 32  01454            .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                3D 3C 02  0145C            .ASCII  <2>\<=\
                                   00 03  0145F P.AJM:     .BYTE   3, 0
                                     0015 01461            .WORD   21
                00 00 00 00 00 00  32 32  01463            .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                3C 3D 02  0146B            .ASCII  <2>\=<\
                                   00 03  0146E P.AJN:     .BYTE   3, 0
                                     000F 01470            .WORD   15
                00 00 00 00 00 00  32 32  01472            .BYTE   50, 50, 0, 0, 0, 0, 0, 0
```

```
                                                    3E  01   0147A             .ASCII  <1>\>\
                                                    00  03   0147C P.AJO:      .BYTE   3, 0
                                                      0011   0147E             .WORD   17
                                00  00  00  00  00  00  32  32 01480           .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                                        3D  3E  02 01488        .ASCII  <2>\>=\
                                                    00  03   0148B P.AJP:      .BYTE   3, 0
                                                      0011   0148D             .WORD   17
                                00  00  00  00  00  00  32  32 0148F           .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                                        3E  3D  02 01497        .ASCII  <2>\=>\
                                                    00  03   0149A P.AJQ:      .BYTE   3, 0
                                                      000D   0149C             .WORD   13
                                00  00  00  00  00  00  32  32 0149E           .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                                    3D  01   014A6             .ASCII  <1>\=\
                                                    00  03   014A8 P.AJR:      .BYTE   3, 0
                                                      000E   014AA             .WORD   14
                                00  00  00  00  00  00  32  32 014AC           .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                                        3E  3C  02 014B4        .ASCII  <2>\<>\
                                                    00  03   014B7 P.AJS:      .BYTE   3, 0
                                                      000E   014B9             .WORD   14
                                00  00  00  00  00  00  32  32 014BB           .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                                        3C  3E  02 014C3        .ASCII  <2>\><\
                                                           014C6             .BLKB   2
                                                  00000017 014C8             .LONG   23
00001340  00001332  00001324  00001315  00001307  000012F9 014CC P.AIV:      .LONG   4857, 4871, 4885, 4900, 4914, 4928, 4942, -
00001395  00001387  00001379  0000136A  0000135C  0000134E 014E4                     4956, 4970, 4985, 4999, 5013, 5027, 5041, -
000013EB  000013DC  000013CD  000013BF  000013B1  000013A3 014FC                     5055, 5069, 5084, 5099, 5113, 5128, 5143, -
          00001434  00001425  00001417  00001408  000013F9 01514                     5157, 5172
                                                    01  00   01528 P.AJU:      .BYTE   0, 1
                                                      0002   0152A             .WORD   2
                                          00  00    00  00   0152C             .BYTE   0, 0, 0, 0
                                                    29  01   01530             .ASCII  <1>\)\
                                                    02  00   01532 P.AJV:      .BYTE   0, 2
                                                      0003   01534             .WORD   3
                                          00  00    00  00   01536             .BYTE   0, 0, 0, 0
                                                    3A  01   0153A             .ASCII  <1>\:\
                                                    00  00   0153C P.AJW:      .BYTE   0, 0
                                                      0001   0153E             .WORD   1
                                          00  00    00  00   01540             .BYTE   0, 0, 0, 0
                                                    2C  01   01544             .ASCII  <1>\,\
                                                           01546             .BLKB   2
                                                  00000003 01548             .LONG   3
000014B9  000014AF  000014A5  0154C P.AJT:      .LONG   5285, 5295, 5305
                                                  00000000 01558             .LONG   0
                                                           0155C P.AJX:      .BLKB   0
                                                  00000000 0155C             .LONG   0
                                                           01560 P.AJY:      .BLKB   0
                                                       02  01560 P.AJZ:      .BYTE   2
                                                       01  01561             .BYTE   1
                                                     0007  01562             .WORD   7
                                                       03  01564             .BYTE   3
                                                       03  01565             .BYTE   3
                                                     0009  01566             .WORD   9
                                                       09  01568             .BYTE   9
                                                       0F  01569             .BYTE   15
                                                     0009  0156A             .WORD   9
                                                       05  0156C             .BYTE   5
                                                       04  0156D             .BYTE   4
```

```
                                                    000F  0156E           .WORD   15
                                                      04  01570           .BYTE   4
                                                      07  01571           .BYTE   7
                                                    0013  01572           .WORD   19
                                                      01  01574           .BYTE   1
                                                      0C  01575           .BYTE   12
                                                    0017  01576           .WORD   23
                                                00000000  01578           .LONG   0
                                                      01  0157C           .BYTE   1
                                                      02  0157D           .BYTE   2
                                                    0017  0157E           .WORD   23
                                                00000000  01580           .LONG   0
                                                      03  01584           .BYTE   3
                                                      0E  01585           .BYTE   14
                                                    0009  01586           .WORD   9
                                                      09  01588           .BYTE   9
                                                      0F  01589           .BYTE   15
                                                    0009  0158A           .WORD   9
                                                      05  0158C           .BYTE   5
                                                      10  0158D           .BYTE   16
                                                    000F  0158E           .WORD   15
                                                      04  01590           .BYTE   4
                                                      12  01591           .BYTE   18
                                                    0013  01592           .WORD   19
                                                      01  01594           .BYTE   1
                                                      17  01595           .BYTE   23
                                                    0017  01596           .WORD   23
                                                00000000  01598           .LONG   0
                                                      05  0159C           .BYTE   5
                                                      19  0159D           .BYTE   25
                                                    000F  0159E           .WORD   15
                                                      04  015A0           .BYTE   4
                                                      1C  015A1           .BYTE   28
                                                    0013  015A2           .WORD   19
                                                      01  015A4           .BYTE   1
                                                      21  015A5           .BYTE   33
                                                    0017  015A6           .WORD   23
                                                00000000  015A8           .LONG   0
                                                      05  015AC           .BYTE   5
                                                      24  015AD           .BYTE   36
                                                    000F  015AE           .WORD   15
                                                      04  015B0           .BYTE   4
                                                      26  015B1           .BYTE   38
                                                    0013  015B2           .WORD   19
                                                      01  015B4           .BYTE   1
                                                      2A  015B5           .BYTE   42
                                                    0017  015B6           .WORD   23
                                                00000000  015B8           .LONG   C
                                                00000000  015BC           .LONG   0
000014C9  000014DD  00000C8D  00001449  000012E1  00001261  015C0 P.AKA:    .LONG   4705, 4833, 5193, 3213, 5341, 5321, 5337, -
00000000  00000000  00000001  00000000  000014DD  000014D9  015D8                   5341, 0, 1, 0, 0, 1
                                                00000001  015F0
                                                00000007  015F4           .LONG   7
5E010800  5D040000  5B000800  3E040000  3C008800  24000007  015F8 P.AKB:    .LONG   603979783, 1006667776, 1040449536, -
                                                5F000007  01610                   1526728704, 1560543232, 1577125888, -
                                                                                  1593835527
                                                  00  03  01614 P.AKD:    .BYTE   3, 0
```

```
                                 0025  01616                .WORD   37
00  00  00  00  00  00  46  46   01618                .BYTE   70, 70, 0, 0, 0, 0, 0, 0
                    44  4F  4D  03   01620                .ASCII  <3>\MOD\
                            00  03   01624 P.AKE:       .BYTE   3, 0
                                 000D  01626                .WORD   13
00  00  00  00  00  00  32  32   01628                .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                    4C  51  45  03   01630                .ASCII  <3>\EQL\
                            00  03   01634 P.AKF:       .BYTE   3, 0
                                 000D  01636                .WORD   13
00  00  00  00  00  00  32  32   01638                .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                55  4C  51  45  04   01640                .ASCII  <4>\EQLU\
                            00  03   01645 P.AKG:       .BYTE   3, 0
                                 000D  01647                .WORD   13
00  00  00  00  00  00  32  32   01649                .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                41  4C  51  45  04   01651                .ASCII  <4>\EQLA\
                            00  03   01656 P.AKH:       .BYTE   3, 0
                                 000E  01658                .WORD   14
00  00  00  00  00  00  32  32   0165A                .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                    51  45  4E  03   01662                .ASCII  <3>\NEQ\
                            00  03   01666 P.AKI:       .BYTE   3, 0
                                 000E  01668                .WORD   14
00  00  00  00  00  00  32  32   0166A                .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                55  51  45  4E  04   01672                .ASCII  <4>\NEQU\
                            00  03   01677 P.AKJ:       .BYTE   3, 0
                                 000E  01679                .WORD   14
00  00  00  00  00  00  32  32   0167B                .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                41  51  45  4E  04   01683                .ASCII  <4>\NEQA\
                            00  03   01688 P.AKK:       .BYTE   3, 0
                                 000F  0168A                .WORD   15
00  00  00  00  00  00  32  32   0168C                .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                    52  54  47  03   01694                .ASCII  <3>\GTR\
                            00  03   01698 P.AKL:       .BYTE   3, 0
                                 0010  0169A                .WORD   16
00  00  00  00  00  00  32  32   0169C                .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                55  52  54  47  04   016A4                .ASCII  <4>\GTRU\
                            00  03   016A9 P.AKM:       .BYTE   3, 0
                                 0010  016AB                .WORD   16
00  00  00  00  00  00  32  32   016AD                .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                41  52  54  47  04   016B5                .ASCII  <4>\GTRA\
                            00  03   016BA P.AKN:       .BYTE   3, 0
                                 0011  016BC                .WORD   17
00  00  00  00  00  00  32  32   016BE                .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                    51  45  47  03   016C6                .ASCII  <3>\GEQ\
                            00  03   016CA P.AKO:       .BYTE   3, 0
                                 0012  016CC                .WORD   18
00  00  00  00  00  00  32  32   016CE                .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                55  51  45  47  04   016D6                .ASCII  <4>\GEQU\
                            00  03   016DB P.AKP:       .BYTE   3, 0
                                 0012  016DD                .WORD   18
00  00  00  00  00  00  32  32   016DF                .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                41  51  45  47  04   016E7                .ASCII  <4>\GEQA\
                            00  03   016EC P.AKQ:       .BYTE   3, 0
                                 0013  016EE                .WORD   19
00  00  00  00  00  00  32  32   016F0                .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                    53  53  4C  03   016F8                .ASCII  <3>\LSS\
                            00  03   016FC P.AKR:       .BYTE   3, 0
                                 0014  016FE                .WORD   20
```

```
                     00   00   00   00   00   00   32   32   01700             .BYTE    50, 50, 0, 0, 0, 0, 0, 0          ;
                                         55   53   53   4C   04   01708             .ASCII   <4>\LSSU\                         ;
                                                   00   03   0170D  P.AKS:     .BYTE    3, 0                              ;
                                                     0014    0170F             .WORD    20                                ;
                     00   00   00   00   00   00   32   32   01711             .BYTE    50, 50, 0, 0, 0, 0, 0, 0          ;
                                         41   53   53   4C   04   01719             .ASCII   <4>\LSSA\                         ;
                                                   00   03   0171E  P.AKT:     .BYTE    3, 0                              ;
                                                     0015    01720             .WORD    21                                ;
                     00   00   00   00   00   00   32   32   01722             .BYTE    50, 50, 0, 0, 0, 0, 0, 0          ;
                                              51   45   4C   03   0172A             .ASCII   <3>\LEQ\                          ;
                                                   00   03   0172E  P.AKU:     .BYTE    3, 0                              ;
                                                     0016    01730             .WORD    22                                ;
                     00   00   00   00   00   00   32   32   01732             .BYTE    50, 50, 0, 0, 0, 0, 0, 0          ;
                                         55   51   45   4C   04   0173A             .ASCII   <4>\LEQU\                         ;
                                                   00   03   0173F  P.AKV:     .BYTE    3, 0                              ;
                                                     0016    01741             .WORD    22                                ;
                     00   00   00   00   00   00   32   32   01743             .BYTE    50, 50, 0, 0, 0, 0, 0, 0          ;
                                         41   51   45   4C   04   0174B             .ASCII   <4>\LEQA\                         ;
                                                   00   02   01750  P.AKW:     .BYTE    2, 0                              ;
                                                     001E    01752             .WORD    30                                ;
                     00   00   00   00   00   00   C8   28   01754             .BYTE    40, -56, 0, 0, 0, 0, 0, 0         ;
                                              54   4F   4E   03   0175C             .ASCII   <3>\NOT\                          ;
                                                   00   03   01760  P.AKX:     .BYTE    3, 0                              ;
                                                     001F    01762             .WORD    31                                ;
                     00   00   00   00   00   00   1E   1E   01764             .BYTE    30, 30, 0, 0, 0, 0, 0, 0          ;
                                              44   4E   41   03   0176C             .ASCII   <3>\AND\                          ;
                                                   00   03   01770  P.AKY:     .BYTE    3, 0                              ;
                                                     0020    01772             .WORD    32                                ;
                     00   00   00   00   00   00   14   14   01774             .BYTE    20, 20, 0, 0, 0, 0, 0, 0          ;
                                                   52   4F   02   0177C             .ASCII   <2>\OR\                           ;
                                                   00   03   0177F  P.AKZ:     .BYTE    3, 0                              ;
                                                     0022    01781             .WORD    34                                ;
                     00   00   00   00   00   00   0A   0A   01783             .BYTE    10, 10, 0, 0, 0, 0, 0, 0          ;
                                              56   51   45   03   0178B             .ASCII   <3>\EQV\                          ;
                                                   00   03   0178F  P.ALA:     .BYTE    3, 0                              ;
                                                     0021    01791             .WORD    33                                ;
                     00   00   00   00   00   00   0A   0A   01793             .BYTE    10, 10, 0, 0, 0, C, 0, 0          ;
                                              52   4F   58   03   0179B             .ASCII   <3>\XOR\                          ;
                                                             0179F             .BLKB    1
                                                 00000018    017A0             .LONG    24                                ;
000015E3  000015D3  000015C2  000015B1  000015A1  00001591    017A4  P.AKC:     .LONG    5521, 5537, 5553, 5570, 5587, 5603, 5620, - ;
00001647  00001637  00001626  00001615  00001605  000015F4    017BC                      5637, 5653, 5670, 5687, 5703, 5720, 5737, - ;
000016AB  0000169B  0000168A  00001679  00001669  00001658    017D4                      5753, 5770, 5787, 5803, 5820, 5837, 5853, - ;
0000170C  000016FC  000016ED  000016DD  000016CD  000016BC    017EC                      5869, 5884, 5900                  ;
                                                   01   02   01804  P.ALC:     .BYTE    2, 1                              ;
                                                     0002    01806             .WORD    2                                 ;
                     00   00   00   00   uu   00   00   00   01808             .BYTE    0, 0, 0, 0, 0, 0, 0, 0            ;
                                                   5C   01   01810             .ASCII   <1><92>                           ;
                                                   01   03   01812  P.ALD:     .BYTE    3, 1                              ;
                                                     0003    01814             .WORD    3                                 ;
                     00   00   00   00   00   00   00   00   01816             .BYTE    0, 0, 0, 0, 0, 0, 0, 0            ;
                                                   5C   01   0181E             .ASCII   <1><92>                           ;
                                                   01   04   01820  P.ALE:     .BYTE    4, 1                              ;
                                                     0004    01822             .WORD    4                                 ;
                     00   00   00   00   00   00   00   00   01824             .BYTE    0, 0, 0, 0, 0, 0, 0, 0            ;
                                                   5B   01   0182C             .ASCII   <1>\[\                            ;
                                                   02   02   0182E  P.ALF:     .BYTE    2, 2                              ;
```

```
                                    000B  01830           .WORD   11
00  00  00  00  00  00  C8  05      01832           .BYTE   5, -56, 0, 0, 0, 0, 0, 0
                        28  01      0183A           .ASCII  <1>\(\
                        02  04      0183C P.ALG:    .BYTE   4, 2
                                    000C  0183E           .WORD   12
00  00  00  00  00  00  06  C8      01840           .BYTE   -56, 6, 0, 0, 0, 0, 0, 0
                        29  01      01848           .ASCII  <1>\)\
                        02  04      0184A P.ALH:    .BYTE   4, 2
                                    0031  0184C           .WORD   49
00  00  00  00  00  00  6E  C8      0184E           .BYTE   -56, 110, 0, 0, 0, 0, 0, 0
                        3C  01      01856           .ASCII  <1>\<\
                        00  02      01858 P.ALI:    .BYTE   2, 0
                                    0003  0185A           .WORD   3
00  00  00  00  00  00  C8  64      0185C           .BYTE   100, -56, 0, 0, 0, 0, 0, 0
                        2E  01      01864           .ASCII  <1>\.\
                        00  02      01866 P.ALJ:    .BYTE   2, 0
                                    0004  01868           .WORD   4
00  00  00  00  00  00  C8  5A      0186A           .BYTE   90, -56, 0, 0, 0, 0, 0, 0
                        2B  01      01872           .ASCII  <1>\+\
                        00  02      01874 P.ALK:    .BYTE   2, 0
                                    0005  01876           .WORD   5
00  00  00  00  00  00  C8  5A      01878           .BYTE   90, -56, 0, 0, 0, 0, 0, 0
                        2D  01      01880           .ASCII  <1>\-\
                        00  03      01882 P.ALL:    .BYTE   3, 0
                                    0026  01884           .WORD   38
00  00  00  00  00  00  50  50      01886           .BYTE   80, 80, 0, 0, 0, 0, 0, 0
                        5E  01      0188E           .ASCII  <1>\^\
                        00  03      01890 P.ALM:    .BYTE   3, 0
                                    0008  01892           .WORD   8
00  00  00  00  00  00  46  46      01894           .BYTE   70, 70, 0, 0, 0, 0, 0, 0
                        2A  01      0189C           .ASCII  <1>\*\
                        00  03      0189E P.ALN:    .BYTE   3, 0
                                    0009  018A0           .WORD   9
                    00  00  00  00  00  00  46  46  018A2           .BYTE   70, 70, 0, 0, 0, 0, 0, 0
                                    2F  01  018AA           .ASCII  <1>\/\
                                    00  03  018AC P.ALO:    .BYTE   3, 0
                                    0006  018AE           .WORD   6
                    00  00  00  00  00  00  3C  3C  018B0           .BYTE   60, 60, 0, 0, 0, 0, 0, 0
                                    2B  01  018B8           .ASCII  <1>\+\
                                    00  03  018BA P.ALP:    .BYTE   3, 0
                                    0007  018BC           .WORD   7
                    00  00  00  00  00  00  3C  3C  018BE           .BYTE   60, 60, 0, 0, 0, 0, 0, 0
                                    2D  01  018C6           .ASCII  <1>\-\
                                0000000E  018C8           .LONG   14
000017C7  000017B9  000017AB  0000179D  0000178F  00001781  018CC P.ALB:    .LONG   6017, 6031, 6045, 6059, 6073, 6087, 6101, -
00001818  0000180D  000017FF  000017F1  000017E3  000017D5  018E4                   6115, 6129, 6143, 6157, 6171, 6185, 6199
                                    00001837  00001829  018FC
                                    00  00  01904 P.ALR:    .BYTE   0, 0
                                    0002  01906           .WORD   2
                        00  00  00  00  01908           .BYTE   0, 0, 0, 0
                                    5D  01  0190C           .ASCII  <1>\]\
                                    02  00  0190E P.ALS:    .BYTE   0, 2
                                    0003  01910           .WORD   3
                        00  00  00  00  01912           .BYTE   0, 0, 0, 0
                                    3A  01  01916           .ASCII  <1>\:\
                                    00  00  01918 P.ALT:    .BYTE   0, 0
                                    0001  0191A           .WORD   1
```

```
                                    00  00  00  00  0191C           .BYTE   0, 0, 0, 0                                      ;
                                        2C  01  01920               .ASCII  <1>\,\                                          ;
                                                01922               .BLKB   2                                               ;
                                    00000003    01924               .LONG   3                                               ;
00001895  0000188B  00001881        01928 P.ALQ:    .LONG   6273, 6283, 6293                                                ;
                    00000000        01934           .LONG   0                                                               ;
                                    01938 P.ALU:    .BLKB   0                                                               ;
                    00000000        01938           .LONG   0                                                               ;
                                    0193C P.ALV:    .BLKB   0                                                               ;
                          02        0193C P.ALW:    .BYTE   2                                                               ;
                          01        0193D           .BYTE   1                                                               ;
                        0006        0193E           .WORD   6                                                               ;
                          03        01940           .BYTE   3                                                               ;
                          03        01941           .BYTE   3                                                               ;
                        0008        01942           .WORD   8                                                               ;
                          09        01944           .BYTE   9                                                               ;
                          0F        01945           .BYTE   15                                                              ;
                        0008        01946           .WORD   8                                                               ;
                          04        01948           .BYTE   4                                                               ;
                          08        01949           .BYTE   8                                                               ;
                        000D        0194A           .WORD   13                                                              ;
                          01        0194C           .BYTE   1                                                               ;
                          0C        0194D           .BYTE   12                                                              ;
                        000F        0194E           .WORD   15                                                              ;
                    00000000        01950           .LONG   0                                                               ;
                          01        01954           .BYTE   1                                                               ;
                          02        01955           .BYTE   2                                                               ;
                        000F        01956           .WORD   15                                                              ;
                    00000000        01958           .LONG   0                                                               ;
                          03        0195C           .BYTE   3                                                               ;
                          0E        0195D           .BYTE   14                                                              ;
                        0008        0195E           .WORD   8                                                               ;
                          09        01960           .BYTE   9                                                               ;
                          0F        01961           .BYTE   15                                                              ;
                        0008        01962           .WORD   8                                                               ;
                          04        01964           .BYTE   4                                                               ;
                          14        01965           .BYTE   20                                                              ;
                        000D        01966           .WORD   13                                                              ;
                          01        01968           .BYTE   1                                                               ;
                          17        01969           .BYTE   23                                                              ;
                        000F        0196A           .WORD   15                                                              ;
                    00000000        0196C           .LONG   0                                                               ;
                          01        01970           .BYTE   1                                                               ;
                          2A        01971           .BYTE   42                                                              ;
                        000F        01972           .WORD   15                                                              ;
                    00000000        01974           .LONG   0                                                               ;
                    00000000        01978           .LONG   0                                                               ;
000018A5  000018B9  00000C8D  00001849  00001721  00001575  0197C P.ALX:    .LONG   5493, 5921, 6217, 3213, 6329, 6309, 6325, -     ;
00000000  00000000  00000001  00000000  000018B9  000018B5  01994                   6329, 0, 1, 0, 0, 0                         ;
                                                  00000000  019AC                                                           ;
                                                  0000000D  019B0           .LONG   13                                      ;
3C00A800  25000800  2A008800  21002800  5F000007  24000007  019B4 P.ALY:    .LONG   603979783, 1593835527, 553658368, -             ;
7C002800  5E010800  5D040000  5B000800  3E042800  3D042800  019CC                   704677888, 620759040, 1006675968, -         ;
                                                  7E000800  019E4                   1023682560, 1040459776, 1526728704, -       ;
                                                                                    1560543232, 1577125888, 2080385024, -       ;
                                                                                    2113931264                                  ;
                                                      00  02  019E8 P.AMA:    .BYTE   2, 0                                    ;
```

```
                                    0029  019EA                .WORD   41
00  00  00  00  00  00  C8  8C      0019EC                     .BYTE   -116, -56, 0, 0, 0, 0, 0, 0
    46  4F  45  5A  49  53  06      019F4                      .ASCII  <6>\SIZEOF\
                                    019FB                      .BLKB   1
                                00000001  019FC                .LONG   1
                                00001965  01A00 P.ALZ:         .LONG   6501
                                  01  02  01A04 P.AMC:         .BYTE   2, 1
                                    0002  01A06                .WORD   2
00  00  00  00  00  00  00  00          01A08                .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                  5C  01  01A10                .ASCII  <1><92>
                                  01  03  01A12 P.AMD:         .BYTE   3, 1
                                    0003  01A14                .WORD   3
00  00  00  00  00  00  00  00          01A16                .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                  5C  01  01A1E                .ASCII  <1><92>
                                  01  04  01A20 P.AME:         .BYTE   4, 1
                                    0004  01A22                .WORD   4
00  00  00  00  00  00  00  00          01A24                .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                  5B  01  01A2C                .ASCII  <1>\[\
                                  01  03  01A2E P.AMF:         .BYTE   3, 1
                                    0005  01A30                .WORD   5
00  00  00  00  00  00  00  00          01A32                .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                  2E  01  01A3A                .ASCII  <1>\.\
                                  02  02  01A3C P.AMG:         .BYTE   2, 2
                                    000B  01A3E                .WORD   11
00  00  00  00  00  00  C8  05          01A40                .BYTE   5, -56, 0, 0, 0, 0, 0, 0
                                  28  01  01A48                .ASCII  <1>\(\
                                  02  04  01A4A P.AMH:         .BYTE   4, 2
                                    000C  01A4C                .WORD   12
00  00  00  00  00  00  06  C8          01A4E                .BYTE   -56, 6, 0, 0, 0, 0, 0, 0
                                  29  01  01A56                .ASCII  <1>\)\
                                  00  02  01A58 P.AMI:         .BYTE   2, 0
                                    0017  01A5A                .WORD   23
00  00  00  00  00  00  C8  8C          01A5C                .BYTE   -116, -56, 0, 0, 0, 0, 0, 0
                                  21  01  01A64                .ASCII  <1>\!\
                                  00  02  01A66 P.AMJ:         .BYTE   2, 0
                                    001E  01A68                .WORD   30
00  00  00  00  00  00  C8  8C          01A6A                .BYTE   -116, -56, 0, 0, 0, 0, 0, 0
                                  7E  01  01A72                .ASCII  <1>\~\
                                  00  02  01A74 P.AMK:         .BYTE   2, 0
                                    0003  01A76                .WORD   3
00  00  00  00  00  00  C8  8C          01A78                .BYTE   -116, -56, 0, 0, 0, 0, 0, 0
                                  2A  01  01A80                .ASCII  <1>\*\
                                  00  03  01A82 P.AML:         .BYTE   3, 0
                                    0008  01A84                .WORD   8
00  00  00  00  00  00  82  82          01A86                .BYTE   -126, -126, 0, 0, 0, 0, 0, 0
                                  2A  01  01A8E                .ASCII  <1>\*\
                                  00  03  01A90 P.AMM:         .BYTE   3, 0
                                    0009  01A92                .WORD   9
00  00  00  00  00  00  82  82          01A94                .BYTE   -126, -126, 0, 0, 0, 0, 0, 0
                                  2F  01  01A9C                .ASCII  <1>\/\
                                  00  03  01A9E P.AMN:         .BYTE   3, 0
                                    0025  01AA0                .WORD   37
00  00  00  00  00  00  82  82          01AA2                .BYTE   -126, -126, 0, 0, 0, 0, 0, 0
                                  25  01  01AAA                .ASCII  <1>\%\
                                  00  03  01AAC P.AMO:         .BYTE   3, 0
                                    0026  01AAE                .WORD   38
00  00  00  00  00  00  6E  6E          01AB0                .BYTE   110, 110, 0, 0, 0, 0, 0, 0
```

```
                                    3C  3C  02  01AB8           .ASCII  <2>\<<\
                                        00  03  01ABB P.AMP:    .BYTE   3, 0
                                          0027  01ABD           .WORD   39
00  00  00  00  00  00  6E  6E  01ABF           .BYTE   110, 110, 0, 0, 0, 0, 0, 0
                                    3E  3E  02  01AC7           .ASCII  <2>\>>\
                                        00  03  01ACA P.AMQ:    .BYTE   3, 0
                                          0013  01ACC           .WORD   19
00  00  00  00  00  00  64  64  01ACE           .BYTE   100, 100, 0, 0, 0, 0, 0, 0
                                    3C  01  01AD6           .ASCII  <1>\<\
                                        00  03  01AD8 P.AMR:    .BYTE   3, 0
                                          0015  01ADA           .WORD   21
00  00  00  00  00  00  64  64  01ADC           .BYTE   100, 100, 0, 0, 0, 0, 0, 0
                                    3D  3C  02  01AE4           .ASCII  <2>\<=\
                                        00  03  01AE7 P.AMS:    .BYTE   3, 0
                                          000F  01AE9           .WORD   15
00  00  00  00  00  00  64  64  01AEB           .BYTE   100, 100, 0, 0, 0, 0, 0, 0
                                    3E  01  01AF3           .ASCII  <1>\>\
                                        00  03  01AF5 P.AMT:    .BYTE   3, 0
                                          0011  01AF7           .WORD   17
00  00  00  00  00  00  64  64  01AF9           .BYTE   100, 100, 0, 0, 0, 0, 0, 0
                                    3D  3E  02  01B01           .ASCII  <2>\>=\
                                        00  03  01B04 P.AMU:    .BYTE   3, 0
                                          000D  01B06           .WORD   13
00  00  00  00  00  00  5A  5A  01B08           .BYTE   90, 90, 0, 0, 0, 0, 0, 0
                                    3D  3D  02  01B10           .ASCII  <2>\==\
                                        00  03  01B13 P.AMV:    .BYTE   3, 0
                                          000E  01B15           .WORD   14
00  00  00  00  00  00  5A  5A  01B17           .BYTE   90, 90, 0, 0, 0, 0, 0, 0
                                    3D  21  02  01B1F           .ASCII  <2>\!=\
                                        00  03  01B22 P.AMW:    .BYTE   3, 0
                                          0021  01B24           .WORD   33
00  00  00  00  00  00  46  46  01B26           .BYTE   70, 70, 0, 0, 0, 0, 0, 0
                                    5E  01  01B2E           .ASCII  <1>\^\
                                        00  03  01B30 P.AMX:    .BYTE   3, 0
                                          0020  01B32           .WORD   32
00  00  00  00  00  00  3C  3C  01B34           .BYTE   60, 60, 0, 0, 0, 0, 0, 0
                                    7C  01  01B3C           .ASCII  <1>\|\
                                        00  03  01B3E P.AMY:    .BYTE   3, 0
                                          001D  01B40           .WORD   29
00  00  00  00  00  00  28  28  01B42           .BYTE   40, 40, 0, 0, 0, 0, 0, 0
                                    7C  7C  02  01B4A           .ASCII  <2>\||\
                                                01B4D           .BLKB   3
                                      00000017  01B50           .LONG   23
000019C7  000019B9  000019AB  0000199D  0000198F  00001981  01B54 P.AMB:  .LONG   6529, 6543, 6557, 6571, 6585, 6599, 6613, -
00001A1B  00001A0D  000019FF  000019F1  000019E3  000019D5  01B6C           6627, 6641, 6655, 6669, 6683, 6697, 6712, -
00001A72  00001A64  00001A55  00001A47  00001A38  00001A29  01B84           6727, 6741, 6756, 6770, 6785, 6800, 6815, -
          00001ABB  00001AAD  00001A9F  00001A90  00001A81  01B9C           6829, 6843
                                        00  02  01BB0 P.AMZ:    .BYTE   2, 0
                                          0028  01BB2           .WORD   40
00  00  00  00  00  00  C8  8C  01BB4           .BYTE   -116, -56, 0, 0, 0, 0, 0, 0
                                    26  01  01BBC           .ASCII  <1>\&\
                                        00  03  01BBE P.ANA:    .BYTE   3, 0
                                          001F  01BC0           .WORD   31
00  00  00  00  00  00  50  50  01BC2           .BYTE   80, 80, 0, 0, 0, 0, 0, 0
                                    26  01  01BCA           .ASCII  <1>\&\
                                        00  03  01BCC P.ANB:    .BYTE   3, 0
                                          001C  01BCE           .WORD   28
```

```
                00 00 00 00 00 00 32 32  01BD0           .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                  26 26  01BD8           .ASCII  <2>\&&\
                                  00 03  01BDB P.ANC:    .BYTE   3, 0
                                     0006 01BDD          .WORD   6
                00 00 00 00 00 00 78 78  01BDF           .BYTE   120, 120, 0, 0, 0, 0, 0, 0
                                  2B 01  01BE7           .ASCII  <1>\+\
                                  00 02  01BE9 P.AND:    .BYTE   2, 0
                                     0005 01BEB          .WORD   5
                00 00 00 00 00 00 C8 8C  01BED           .BYTE   -116, -56, 0, 0, 0, 0, 0, 0
                                  2D 01  01BF5           .ASCII  <1>\-\
                                  00 03  01BF7 P.ANE:    .BYTE   3, 0
                                     0007 01BF9          .WORD   7
                00 00 00 00 00 00 78 78  01BFB           .BYTE   120, 120, 0, 0, 0, 0, 0, 0
                                  2D 01  01C03           .ASCII  <1>\-\
                                  01 03  01C05 P.ANF:    .BYTE   3, 1
                                     0008 01C07          .WORD   8
                00 00 00 00 00 00 00 00  01C09           .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                  3E 2D  01C11           .ASCII  <2>\->\
                                  00 02  01C14 P.ANG:    .BYTE   2, 0
                                     0003 01C16          .WORD   3
                00 00 00 00 00 00 C8 28  01C18           .BYTE   40, -56, 0, 0, 0, 0, 0, 0
                                  2A 01  01C20           .ASCII  <1>\*\
                                  00 00  01C22 P.ANI:    .BYTE   0, 0
                                     0002 01C24          .WORD   2
                            00 00 00 00  01C26           .BYTE   0, 0, 0, 0
                                  5D 01  01C2A           .ASCII  <1>\]\
                                  02 00  01C2C P.ANJ:    .BYTE   0, 2
                                     0003 01C2E          .WORD   3
                            00 00 00 00  01C30           .BYTE   0, 0, 0, 0
                                  3A 01  01C34           .ASCII  <1>\:\
                                  00 00  01C36 P.ANK:    .BYTE   0, 0
                                     0001 01C38          .WORD   1
                         00 00 00 00 00  01C3A           .BYTE   0, 0, 0, 0
                                  2C 01  01C3E           .ASCII  <1>\,\
                               00000003  01C40           .LONG   3
00001BB3  00001BA9             00001B9F  01C44 P.ANH:    .LONG   7071, 7081, 7091
                               00000000  01C50           .LONG   0
                                         01C54 P.ANL:    .BLKB   0
                               00000000  01C54           .LONG   0
                                         01C58 P.ANM:    .BLKB   0
                                     02  01C58 P.ANN:    .BYTE   2
                                     01  01C59           .BYTE   1
                                   0008  01C5A           .WORD   8
                                     03  01C5C           .BYTE   3
                                     03  01C5D           .BYTE   3
                                   000A  01C5E           .WORD   10
                                     09  01C60           .BYTE   9
                                     0F  01C61           .BYTE   15
                                   000A  01C62           .WORD   10
                                     05  01C64           .BYTE   5
                                     04  01C65           .BYTE   4
                                   0011  01C66           .WORD   17
                                     04  01C68           .BYTE   4
                                     07  01C69           .BYTE   7
                                   0016  01C6A           .WORD   22
                                     08  01C6C           .BYTE   8
                                     0A  01C6D           .BYTE   10
```

```
                0011  01C6E          .WORD   17
                  01  01C70          .BYTE   1
                  0C  01C71          .BYTE   12
                001B  01C72          .WORD   27
            00000000  01C74          .LONG   0
                  01  01C78          .BYTE   1
                  02  01C79          .BYTE   2
                001B  01C7A          .WORD   27
            00000000  01C7C          .LONG   0
                  03  01C80          .BYTE   3
                  0E  01C81          .BYTE   14
                000A  01C82          .WORD   10
                  09  01C84          .BYTE   9
                  0F  01C85          .BYTE   15
                000A  01C86          .WORD   10
                  05  01C88          .BYTE   5
                  10  01C89          .BYTE   16
                0011  01C8A          .WORD   17
                  04  01C8C          .BYTE   4
                  12  01C8D          .BYTE   18
                0016  01C8E          .WORD   22
                  08  01C90          .BYTE   8
                  15  01C91          .BYTE   21
                0011  01C92          .WORD   17
                  01  01C94          .BYTE   1
                  17  01C95          .BYTE   23
                001B  01C96          .WORD   27
            00000000  01C98          .LONG   0
                  05  01C9C          .BYTE   5
                  19  01C9D          .BYTE   25
                0011  01C9E          .WORD   17
                  04  01CA0          .BYTE   4
                  1C  01CA1          .BYTE   28
                0016  01CA2          .WORD   22
                  08  01CA4          .BYTE   8
                  1F  01CA5          .BYTE   31
                0011  01CA6          .WORD   17
                  01  01CA8          .BYTE   1
                  21  01CA9          .BYTE   33
                001B  01CAA          .WORD   27
            00000000  01CAC          .LONG   0
                  05  01CB0          .BYTE   5
                  24  01CB1          .BYTE   36
                0011  01CB2          .WORD   17
                  04  01CB4          .BYTE   4
                  26  01CB5          .BYTE   38
                0016  01CB6          .WORD   22
                  08  01CB8          .BYTE   8
                  28  01CB9          .BYTE   40
                0011  01CBA          .WORD   17
                  01  01CBC          .BYTE   1
                  2A  01CBD          .BYTE   42
                001B  01CBE          .WORD   27
            00000000  01CC0          .LONG   0
            00000000  01CC4          .LONG   0
00001BC1    00001BD5  00000C8D  00001AD1  0000197D  00001931  01CC8 P.ANO:  .LONG   6449, 6525, 6865, 3213, 7125, 7105, 7121, -
00000000    00000001  00000000  00000001  00001BD5  00001BD1  01CE0         .LONG   7125, 1, 0, 1, 0, 0
```

```
                                                  00000000  01CF8
                                                  00000010  01CFC          .LONG   16
340000?1F  3300021F  3200021F  3100021F  300002'F  2D008852  01D00 P.ANP:  .LONG   755009618, 805306911, 822084127, -
3C04A800   3900021F  3800021F  3700021F  3600021F  3500021F  01D18                 838861343, 855638559, 872415775, -
                     24000007  5F000007  3D042800  3E042800  01D30                 889192991, 905970207, 922747423, -
                                                                                   939524639, 956301855, 1006938112, -
                                                                                   1040459776, 1023682560, 1593835527, -
                                                                                   603979783
                                                     01  03  01D40 P.ANR:  .BYTE   3, 1
                                                       0005  01D42          .WORD   5
                                  00  00  00  00  00  00  00  00  01D44          .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                  46  4F  02  01D4C          .ASCII  <2>\OF\
                                                     01  03  01D4F P.ANS:  .BYTE   3, 1
                                                       0005  01D51          .WORD   5
                                  00  00  00  00  00  00  00  00  01D53          .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                  4E  49  02  01D5B          .ASCII  <2>\IN\
                                                     00  02  01D5E P.ANT:  .BYTE   2, 0
                                                       0017  01D60          .WORD   23
                                  00  00  00  00  00  00  C8  19  01D62          .BYTE   25, -56, 0, 0, 0, 0, 0, 0
                                              54  4F  4E  03  01D6A          .ASCII  <3>\NOT\
                                                     00  03  01D6E P.ANU:  .BYTE   3, 0
                                                       0018  01D70          .WORD   24
                                  00  00  00  00  00  00  14  14  01D72          .BYTE   20, 20, 0, 0, 0, 0, 0, 0
                                              44  4E  41  03  01D7A          .ASCII  <3>\AND\
                                                     00  03  01D7E P.ANV:  .BYTE   3, 0
                                                       0019  01D80          .WORD   25
                                  00  00  00  00  00  00  0A  0A  01D82          .BYTE   10, 10, 0, 0, 0, 0, 0, 0
                                                  52  4F  02  01D8A          .ASCII  <2>\OR\
                                                     02  03  01D8D P.ANW:  .BYTE   3, 2
                                                       002C  01D8F          .WORD   44
                                  00  00  00  00  00  00  1E  1E  01D91          .BYTE   30, 30, 0, 0, 0, 0, 0, 0
                                              54  4F  4E  03  01D99          .ASCII  <3>\NOT\
                                                              01D9D          .BLKB   3
                                                  00000006    01DA0          .LONG   6
00001D0A   00001CFB  00001CEB  00001CDB  00001CCC  00001CBD  01DA4 P.ANQ:  .LONG   7357, 7372, 7387, 7403, 7419, 7434
                                                     00  03  01DBC P.ANX:  .BYTE   3, 0
                                                       000E  01DBE          .WORD   14
                                  00  00  00  00  00  00  1E  1E  01DC0          .BYTE   30, 30, 0, 0, 0, 0, 0, 0
                                          3D  20  54  4F  4E  05  01DC8          .ASCII  <5>\NOT =\
                                                     00  03  01DCE P.ANY:  .BYTE   3, 0
                                                       0015  01DD0          .WORD   21
                                  00  00  00  00  00  00  1E  1E  01DD2          .BYTE   30, 30, 0, 0, 0, 0, 0, 0
                                          3E  20  54  4F  4E  05  01DDA          .ASCII  <5>\NOT >\
                                                     00  03  01DE0 P.ANZ:  .BYTE   3, 0
                                                       0011  01DE2          .WORD   17
                                  00  00  00  00  00  00  1E  1E  01DE4          .BYTE   30, 30, 0, 0, 0, 0, 0, 0
                                          3C  20  54  4F  4E  05  01DEC          .ASCII  <5>\NOT <\
                                                     01  02  01DF2 P.AOB:  .BYTE   2, 1
                                                       0002  01DF4          .WORD   2
                                  00  00  00  00  00  00  00  00  01DF6          .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                     5C  01  01DFE          .ASCII  <1><92>
                                                     01  03  01E00 P.AOC:  .BYTE   3, 1
                                                       0003  01E02          .WORD   3
                                  00  00  00  00  00  00  00  00  01E04          .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                     5C  01  01E0C          .ASCII  <1><92>
                                                     01  04  01E0E P.AOD:  .BYTE   4, 1
                                                       0004  01E10          .WORD   4
```

```
00 00 00 00 00 00 00 00  01E12                .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                  28 01  01E1A                .ASCII  <1>\(\
                  02 02  01E1C P.AOE:         .BYTE   2, 2
                   003F  01E1E                .WORD   63
00 00 00 00 00 00 C8 1E  01E20                .BYTE   30, -56, 0, 0, 0, 0, 0, 0
                  3E 01  01E28                .ASCII  <1>\>\
                  02 02  01E2A P.AOF:         .BYTE   2, 2
                   0040  01E2C                .WORD   64
00 00 00 00 00 00 C8 1E  01E2E                .BYTE   30, -56, 0, 0, 0, 0, 0, 0
                  3C 01  01E36                .ASCII  <1>\<\
                  02 02  01E38 P.AOG:         .BYTE   2, 2
                   0041  01E3A                .WORD   65
00 00 00 00 00 00 C8 1E  01E3C                .BYTE   30, -56, 0, 0, 0, 0, 0, 0
                  3D 01  01E44                .ASCII  <1>\=\
                  02 02  01E46 P.AOH:         .BYTE   2, 2
                   000B  01E48                .WORD   11
00 00 00 00 00 00 C8 05  01E4A                .BYTE   5, -56, 0, 0, 0, 0, 0, 0
                  28 01  01E52                .ASCII  <1>\(\
                  02 04  01E54 P.AOI:         .BYTE   4, 2
                   000C  01E56                .WORD   12
00 00 00 00 00 00 06 C8  01E58                .BYTE   -56, 6, 0, 0, 0, 0, 0, 0
                  29 01  01E60                .ASCII  <1>\)\
                  00 03  01E62 P.AOJ:         .BYTE   3, 0
                   0008  01E64                .WORD   8
00 00 00 00 00 00 3C 3C  01E66                .BYTE   60, 60, 0, 0, 0, 0, 0, 0
                  2A 01  01E6E                .ASCII  <1>\*\
                  00 03  01E70 P.AOK:         .BYTE   3, 0
                   0009  01E72                .WORD   9
00 00 00 00 00 00 3C 3C  01E74                .BYTE   60, 60, 0, 0, 0, 0, 0, 0
                  2F 01  01E7C                .ASCII  <1>\/\
                  00 02  01E7E P.AOL:         .BYTE   2, 0
                   0004  01E80                .WORD   4
00 00 00 00 00 00 C8 32  01E82                .BYTE   50, -56, 0, 0, 0, 0, 0, 0
                  2B 01  01E8A                .ASCII  <1>\+\
                  00 02  01E8C P.AOM:         .BYTE   2, 0
                   0005  01E8E                .WORD   5
00 00 00 00 00 00 C8 32  01E90                .BYTE   50, -56, 0, 0, 0, 0, 0, 0
                  2D 01  01E98                .ASCII  <1>\-\
                  00 03  01E9A P.AON:         .BYTE   3, 0
                   0006  01E9C                .WORD   6
00 00 00 00 00 00 28 28  01E9E                .BYTE   40, 40, 0, 0, 0, 0, 0, 0
                  2B 01  01EA6                .ASCII  <1>\+\
                  00 03  01EA8 P.AOO:         .BYTE   3, 0
                   0007  01EAA                .WORD   7
00 00 00 00 00 00 28 28  01EAC                .BYTE   40, 40, 0, 0, 0, 0, 0, 0
                  2D 01  01EB4                .ASCII  <1>\-\
                  00 03  01EB6 P.AOP:         .BYTE   3, 0
                   000F  01EB8                .WORD   15
00 00 00 00 00 00 1E 1E  01EBA                .BYTE   30, 30, 0, 0, 0, 0, 0, 0
                  3E 01  01EC2                .ASCII  <1>\>\
                  00 03  01EC4 P.AOQ:         .BYTE   3, 0
                   0013  01EC6                .WORD   19
00 00 00 00 00 00 1E 1E  01EC8                .BYTE   30, 30, 0, 0, 0, 0, 0, 0
                  3C 01  01ED0                .ASCII  <1>\<\
                  00 03  01ED2 P.AOR:         .BYTE   3, 0
                   000D  01ED4                .WORD   13
00 00 00 00 00 00 1E 1E  01ED6                .BYTE   30, 30, 0, 0, 0, 0, 0, 0
```

```
                                                    3D  01  01EDE           .ASCII  <1>\=\
                                                00000011  01EE0           .LONG   17
00001DB5  00001DA7  00001D99  00001D8B  00001D7D  00001D6F  01EE4 P.AOA:    .LONG   7535, 7549, 7563, 7577, 7591, 7605, 7619, -
00001E09  00001DFB  00001DED  00001DDF  00001DD1  00001DC3  01EFC                   7633, 7647, 7661, 7675, 7689, 7703, 7717, -
          00001E4F  00001E41  00001E33  00001E25  00001E17  01F14                   7731, 7745, 7759
                                                    00  00  01F28 P.AOT:    .BYTE   0, 0
                                                      0002  01F2A           .WORD   2
                                                00  00  00  00  01F2C           .BYTE   0, 0, 0, 0
                                                    29  01  01F30           .ASCII  <1>\)\
                                                    02  00  01F32 P.AOU:    .BYTE   0, 2
                                                      0003  01F34           .WORD   3
                                                00  00  00  00  01F36           .BYTE   0, 0, 0, 0
                                                    3A  01  01F3A           .ASCII  <1>\:\
                                                    00  00  01F3C P.AOV:    .BYTE   0, 0
                                                      0001  01F3E           .WORD   1
                                                00  00  00  00  01F40           .BYTE   0, 0, 0, 0
                                                    2C  01  01F44           .ASCII  <1>\,\
                                                            01F46           .BLKB   2
                                                00000003  01F48           .LONG   3
                                    00001EB9  00001EAF  00001EA5  01F4C P.AOS:    .LONG   7845, 7855, 7865
                                                00000000  01F58           .LONG   0
                                                          01F5C P.AOW:    .BLKB   0
                                                00000000  01F5C           .LONG   0
                                                          01F60 P.AOX:    .BLKB   0
                                                      01  01F60 P.AOY:    .BYTE   1
                                                      0E  01F61           .BYTE   14
                                                    0005  01F62           .WORD   5
                                                      03  01F64           .BYTE   3
                                                      01  01F65           .BYTE   1
                                                    0003  01F66           .WORD   3
                                                      00  01F68           .BYTE   0
                                                      0A  01F69           .BYTE   10
                                                    0021  01F6A           .WORD   33
                                                      01  01F6C           .BYTE   1
                                                      0F  01F6D           .BYTE   15
                                                    0012  01F6E           .WORD   18
                                                      00  01F70           .BYTE   0
                                                      0A  01F71           .BYTE   10
                                                    0021  01F72           .WORD   33
                                                      01  01F74           .BYTE   1
                                                      0E  01F75           .BYTE   14
                                                    0005  01F76           .WORD   5
                                                      03  01F78           .BYTE   3
                                                      01  01F79           .BYTE   1
                                                    0012  01F7A           .WORD   18
                                                      02  01F7C           .BYTE   2
                                                      0D  01F7D           .BYTE   13
                                                    000C  01F7E           .WORD   12
                                                      06  01F80           .BYTE   6
                                                      0D  01F81           .BYTE   13
                                                    000C  01F82           .WORD   12
                                                      07  01F84           .BYTE   7
                                                      0D  01F85           .BYTE   13
                                                    000C  01F86           .WORD   12
                                                      08  01F88           .BYTE   8
                                                      0D  01F89           .BYTE   13
                                                    000C  01F8A           .WORD   12
```

```
                00  01F8C          .BYTE   0
                0C  01F8D          .BYTE   12
              0021  01F8E          .WORD   33
                01  01F90          .BYTE   1
                01  01F91          .BYTE   1
              000C  01F92          .WORD   12
                02  01F94          .BYTE   2
                01  01F95          .BYTE   1
              000C  01F96          .WORD   12
                06  01F98          .BYTE   6
                01  01F99          .BYTE   1
              000C  01F9A          .WORD   12
                07  01F9C          .BYTE   7
                01  01F9D          .BYTE   1
              000C  01F9E          .WORD   12
                08  01FA0          .BYTE   8
                01  01FA1          .BYTE   1
              000C  01FA2          .WORD   12
                00  01FA4          .BYTE   0
                0C  01FA5          .BYTE   12
              0021  01FA6          .WORD   33
                01  01FA8          .BYTE   1
                0F  01FA9          .BYTE   15
              0012  01FAA          .WORD   18
                03  01FAC          .BYTE   3
                08  01FAD          .BYTE   8
              0021  01FAE          .WORD   33
                08  01FB0          .BYTE   8
                05  01FB1          .BYTE   5
              0019  01FB2          .WORD   25
                07  01FB4          .BYTE   7
                06  01FB5          .BYTE   6
              0019  01FB6          .WORD   25
                0D  01FB8          .BYTE   13
                12  01FB9          .BYTE   18
              0019  01FBA          .WORD   25
                09  01FBC          .BYTE   9
                07  01FBD          .BYTE   7
              0019  01FBE          .WORD   25
                00  01FC0          .BYTE   0
                10  01FC1          .BYTE   16
              0021  01FC2          .WORD   33
                01  01FC4          .BYTE   1
                01  01FC5          .BYTE   1
              001F  01FC6          .WORD   31
                04  01FC8          .BYTE   4
                01  01FC9          .BYTE   1
              001D  01FCA          .WORD   29
                05  01FCC          .BYTE   5
                01  01FCD          .BYTE   1
              001D  01FCE          .WORD   29
                00  01FD0          .BYTE   0
                08  01FD1          .BYTE   8
              0021  01FD2          .WORD   33
                01  01FD4          .BYTE   1
                01  01FD5          .BYTE   1
              001F  01FD6          .WORD   31
```

```
                00  01FD8          .BYTE   0
                09  01FD9          .BYTE   9
              0021  01FDA          .WORD   33
                01  01FDC          .BYTE   1
                01  01FDD          .BYTE   1
              001F  01FDE          .WORD   31
                00  01FE0          .BYTE   0
                09  01FE1          .BYTE   9
              0021  01FE2          .WORD   33
                00  01FE4          .BYTE   0
                0B  01FE5          .BYTE   11
              0021  01FE6          .WORD   33
                02  01FE8 P.AOZ:   .BYTE   2
                01  01FE9          .BYTE   1
              0007  01FEA          .WORD   7
                03  01FEC          .BYTE   3
                03  01FED          .BYTE   3
              0009  01FEE          .WORD   9
                09  01FF0          .BYTE   9
                0F  01FF1          .BYTE   15
              0009  01FF2          .WORD   9
                05  01FF4          .BYTE   5
                06  01FF5          .BYTE   6
              000E  01FF6          .WORD   14
                04  01FF8          .BYTE   4
                09  01FF9          .BYTE   9
              0013  01FFA          .WORD   19
                01  01FFC          .BYTE   1
                0C  01FFD          .BYTE   12
              0018  01FFE          .WORD   24
          00000000  02000          .LONG   0
                01  02004          .BYTE   1
                02  02005          .BYTE   2
              0018  02006          .WORD   24
          00000000  02008          .LONG   0
                03  0200C          .BYTE   3
                0E  0200D          .BYTE   14
              0009  0200E          .WORD   9
                09  02010          .BYTE   9
                0F  02011          .BYTE   15
              0009  02012          .WORD   9
                04  02014          .BYTE   4
                13  02015          .BYTE   19
              0013  02016          .WORD   19
                01  02018          .BYTE   1
                17  02019          .BYTE   23
              0018  0201A          .WORD   24
          00000000  0201C          .LONG   0
                03  02020          .BYTE   3
                18  02021          .BYTE   24
              0009  02022          .WORD   9
                05  02024          .BYTE   5
                1B  02025          .BYTE   27
              000E  02026          .WORD   14
                04  02028          .BYTE   4
                1E  02029          .BYTE   30
              0013  0202A          .WORD   19
```

```
                                                    01  0202C           .BYTE   1
                                                    23  0202D           .BYTE   35
                                                0018    0202E           .WORD   24
                                            00000000    02030           .LONG   0
                                                    04  02034           .BYTE   4
                                                    27  02035           .BYTE   39
                                                0016    02036           .WORD   22
                                                    01  02038           .BYTE   1
                                                    2B  02039           .BYTE   43
                                                0018    0203A           .WORD   24
                                            00000000    0203C           .LONG   0
                                                    01  02040           .BYTE   1
                                                    2B  02041           .BYTE   43
                                                0018    02042           .WORD   24
                                            00000000    02044           .LONG   0
                                            00000000    02048           .LONG   0
00001EC9  00001F65  00001EDD  00001E61  00001D21  00001C7D  0204C P.APA:  .LONG   7293, 7457, 7777, 7901, 8037, 7881, 7897, -
00000001  00000000  00000001  00000000  00001EDD  00001ED9  02064           7901, 0, 1, 0, 1, 0
                                            00000000    0207C
                                            00000002    02080           .LONG   2
                                    2F002800    2E010038  02084 P.APB:  .LONG   771817528, 788539392
                                            00000000    0208C           .LONG   0
                                                        02090 P.APC:  .BLKB   0
                                                  01  02  02090 P.APE:  .BYTE   2, 1
                                                0002    02092           .WORD   2
                        00  00  00  00  00  00  00  00   02094           .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                  5C  01  0209C           .ASCII  <1><92>
                                                  01  03  0209E P.APF:  .BYTE   3, 1
                                                0003    020A0           .WORD   3
                        00  00  00  00  00  00  00  00   020A2           .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                  5C  01  020AA           .ASCII  <1><92>
                                                  01  04  020AC P.APG:  .BYTE   4, 1
                                                0004    020AE           .WORD   4
00  00  00  00  00  00  00  00                    00  00  020B0           .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                  28  01  020B8           .ASCII  <1>\(\
                                                  01  03  020BA P.APH:  .BYTE   3, 1
                                                0005    020BC           .WORD   5
00  00  00  00  00  00  00  00                    00  00  020BE           .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                  2E  01  020C6           .ASCII  <1>\.\
                                                  00  03  020C8 P.API:  .BYTE   3, 0
                                                0006    020CA           .WORD   6
00  00  00  00  00  00  3C  3C                    3C  3C  020CC           .BYTE   60, 60, 0, 0, 0, 0, 0, 0
                                                  2B  01  020D4           .ASCII  <1>\+\
                                                  00  03  020D6 P.APJ:  .BYTE   3, 0
                                                0007    020D8           .WORD   7
00  00  00  00  00  00  3C  3C                    3C  3C  020DA           .BYTE   60, 60, 0, 0, 0, 0, 0, 0
                                                  2D  01  020E2           .ASCII  <1>\-\
                                                  00  02  020E4 P.APK:  .BYTE   2, 0
                                                0004    020E6           .WORD   4
00  00  00  00  00  00  C8  46                    C8  46  020E8           .BYTE   70, -56, 0, 0, 0, 0, 0, 0
                                                  2B  01  020F0           .ASCII  <1>\+\
                                                  00  02  020F2 P.APL:  .BYTE   2, 0
                                                0005    020F4           .WORD   5
00  00  00  00  00  00  C8  46                    C8  46  020F6           .BYTE   70, -56, 0, 0, 0, 0, 0, 0
                                                  2D  01  020FE           .ASCII  <1>\-\
                                                  00  03  02100 P.APM:  .BYTE   3, 0
                                                0008    02102           .WORD   8
```

```
                                          00  00  00  00  00  00  50  50  02104           .BYTE   80, 80, 0, 0, 0, 0, 0, 0              ;
                                                                  2A  01  0210C           .ASCII  <1>\*\                                ;
                                                                  00  03  0210E P.APN:    .BYTE   3, 0                                  ;
                                                                    0009  02110           .WORD   9                                     ;
                                          00  00  00  00  00  00  50  50  02112           .BYTE   80, 80, 0, 0, 0, 0, 0, 0              ;
                                                                  2F  01  0211A           .ASCII  <1>\/\                                ;
                                                                  00  03  0211C P.APO:    .BYTE   3, 0                                  ;
                                                                    000A  0211E           .WORD   10                                    ;
                                          00  00  00  00  00  00  5C  5A  02120           .BYTE   90, 92, 0, 0, 0, 0, 0, 0              ;
                                                              2A  2A  02  02128           .ASCII  <2>\**\                               ;
                                                                  00  03  0212B P.APP:    .BYTE   3, 0                                  ;
                                                                    0023  0212D           .WORD   35                                    ;
                                          00  00  00  00  00  00  3C  3C  0212F           .BYTE   60, 60, 0, 0, 0, 0, 0, 0              ;
                                                              2F  2F  02  02137           .ASCII  <2>\//\                               ;
                                                                  02  02  0213A P.APQ:    .BYTE   2, 2                                  ;
                                                                    000B  0213C           .WORD   11                                    ;
                                          00  00  00  00  00  00  C8  05  0213E           .BYTE   5, -56, 0, 0, 0, 0, 0, 0              ;
                                                                  28  01  02146           .ASCII  <1>\(\                                ;
                                                                  02  04  02148 P.APR:    .BYTE   4, 2                                  ;
                                                                    000C  0214A           .WORD   12                                    ;
                                          00  00  00  00  00  00  06  C8  0214C           .BYTE   -56, 6, 0, 0, 0, 0, 0, 0              ;
                                                                  29  01  02154           .ASCII  <1>\)\                                ;
                                                                          02156           .BLKB   2                                     ;
                                                                0000000E  02158           .LONG   14                                    ;
00002053  00002045  00002037  00002029  0000201B  0000200D  0215C P.APD:    .LONG   8205, 8219, 8233, 8247, 8261, 8275, 8289, - ;
000020A8  00002099  0000208B  0000207D  0000206F  00002061  02174                   8303, 8317, 8331, 8345, 8360, 8375, 8389    ;
                                                  000020C5  000020B7  0218C                                                         ;
                                                                  00  02  02194 P.APS:    .BYTE   2, 0                                  ;
                                                                    0003  02196           .WORD   3                                     ;
                                          00  00  00  00  00  00  C8  28  02198           .BYTE   40, -56, 0, 0, 0, 0, 0, 0             ;
                                                                  2E  01  021A0           .ASCII  <1>\.\                                ;
                                                                  01  03  021A2 P.APT:    .BYTE   3, 1                                  ;
                                                                    0005  021A4           .WORD   5                                     ;
                                          00  00  00  00  00  00  00  00  021A6           .BYTE   0, 0, 0, 0, 0, 0, 0, 0                ;
                                                                  2E  01  021AE           .ASCII  <1>\.\                                ;
                                                                  00  03  021B0 P.APV:    .BYTE   3, 0                                  ;
                                                                    000D  021B2           .WORD   13                                    ;
                                          00  00  00  00  00  00  32  32  021B4           .BYTE   50, 50, 0, 0, 0, 0, 0, 0              ;
                                                      2E  51  45  2E  04  021BC           .ASCII  <4>\.EQ.\                             ;
                                                                  00  03  021C1 P.APW:    .BYTE   3, 0                                  ;
                                                                    000E  021C3           .WORD   14                                    ;
                                          00  00  00  00  00  00  32  32  021C5           .BYTE   50, 50, 0, 0, 0, 0, 0, 0              ;
                                                      2E  45  4E  2E  04  021CD           .ASCII  <4>\.NE.\                             ;
                                                                  00  03  021D2 P.APX:    .BYTE   3, 0                                  ;
                                                                    000F  021D4           .WORD   15                                    ;
                                          00  00  00  00  00  00  32  32  021D6           .BYTE   50, 50, 0, 0, 0, 0, 0, 0              ;
                                                      2E  54  47  2E  04  021DE           .ASCII  <4>\.GT.\                             ;
                                                                  00  03  021E3 P.APY:    .BYTE   3, 0                                  ;
                                                                    0011  021E5           .WORD   17                                    ;
                                          00  00  00  00  00  00  32  32  021E7           .BYTE   50, 50, 0, 0, 0, 0, 0, 0              ;
                                                      2E  45  47  2E  04  021EF           .ASCII  <4>\.GE.\                             ;
                                                                  00  03  021F4 P.APZ:    .BYTE   3, 0                                  ;
                                                                    0013  021F6           .WORD   19                                    ;
                                          00  00  00  00  00  00  32  32  021F8           .BYTE   50, 50, 0, 0, 0, 0, 0, 0              ;
                                                      2E  54  4C  2E  04  02200           .ASCII  <4>\.LT.\                             ;
                                                                  00  03  02205 P.AQA:    .BYTE   3, 0                                  ;
                                                                    0015  02207           .WORD   21                                    ;
```

```
                        00  00  00  00  00  00  32  32  02209          .BYTE   50, 50, 0, 0, 0, 0, 0, 0                    ;
                                2E  45  4C  2E  04  02211              .ASCII  <4>\.LE.\                                   ;
                                            00  02  02216 P.AQB:       .BYTE   2, 0                                        ;
                                              0017  02218              .WORD   23                                          ;
                        00  00  00  00  00  00  C8  28  0221A          .BYTE   40, -56, 0, 0, 0, 0, 0, 0                   ;
                                2E  54  4F  4E  2E  05  02222          .ASCII  <5>\.NOT.\                                  ;
                                            00  03  02228 P.AQC:       .BYTE   3, 0                                        ;
                                              0018  0222A              .WORD   24                                          ;
                        00  00  00  00  00  00  1E  1E  0222C          .BYTE   30, 30, 0, 0, 0, 0, 0, 0                    ;
                                2E  44  4E  41  2E  05  02234          .ASCII  <5>\.AND.\                                  ;
                                            00  03  0223A P.AQD:       .BYTE   3, 0                                        ;
                                              0019  0223C              .WORD   25                                          ;
                        00  00  00  00  00  00  14  14  0223E          .BYTE   20, 20, 0, 0, 0, 0, 0, 0                    ;
                                2E  52  4F  2E  04  02246              .ASCII  <4>\.OR.\                                   ;
                                            00  03  0224B P.AQE:       .BYTE   3, 0                                        ;
                                              001A  0224D              .WORD   26                                          ;
                        00  00  00  00  00  00  0A  0A  0224F          .BYTE   10, 10, 0, 0, 0, 0, 0, 0                    ;
                                2E  52  4F  58  2E  05  02257          .ASCII  <5>\.XOR.\                                  ;
                                            00  03  0225D P.AQF:       .BYTE   3, 0                                        ;
                                              001B  0225F              .WORD   27                                          ;
                        00  00  00  00  00  00  0A  0A  02261          .BYTE   10, 10, 0, 0, 0, 0, 0, 0                    ;
                                2E  56  51  45  2E  05  02269          .ASCII  <5>\.EQV.\                                  ;
                                            00  03  0226F P.AQG:       .BYTE   3, 0                                        ;
                                              001A  02271              .WORD   26                                          ;
                        00  00  00  00  00  00  0A  0A  02273          .BYTE   10, 10, 0, 0, 0, 0, 0, 0                    ;
                            2E  56  51  45  4E  2E  06  0227B          .ASCII  <6>\.NEQV.\                                 ;
                                                        02282          .BLKB   2                                           ;
                                                  0000000C  02284      .LONG   12                                          ;
00002182  00002171  00002160  0000214F  0000213E  0000212D  02288 P.APU:  .LONG   8493, 8510, 8527, 8544, 8561, 8578, 8595, -;
000021EC  000021DA  000021C8  000021B7  000021A5  00002193  022A0                  8613, 8631, 8648, 8666, 8684                ;
                                                    01  00  022B8 P.AQI:  .BYTE   0, 1                                       ;
                                                      0002  022BA          .WORD   2                                         ;
                                                00  00  00  00  022BC      .BYTE   0, 0, 0, 0                                ;
                                                    29  01  022C0          .ASCII  <1>\)\                                    ;
                                                    00  00  022C2 P.AQJ:   .BYTE   0, 0                                      ;
                                                      0001  022C4          .WORD   1                                         ;
                                                00  00  00  00  022C6      .BYTE   0, 0, 0, 0                                ;
                                                    2C  01  022CA          .ASCII  <1>\,\                                    ;
                                                    00  00  022CC P.AQK:   .BYTE   0, 0                                      ;
                                                      0003  022CE          .WORD   3                                         ;
                                                00  00  00  00  022D0      .BYTE   0, 0, 0, 0                                ;
                                                    3A  01  022D4          .ASCII  <1>\:\                                    ;
                                                            022D6          .BLKB   2                                         ;
                                                  00000003  022D8          .LONG   3                                         ;
                                00002249  0000223F  00002235  022DC P.AQH:  .LONG   8757, 8767, 8777                        ;
                                                00  02  08  01  022E8 P.AQM:  .BYTE   1, 8, 2, 0                          ;
                                                  00000001  022EC          .LONG   1                                         ;
                                    2E  45  55  52  54  2E  06  022F0      .ASCII  <6>\.TRUE.\                               ;
                                                00  02  08  01  022F7 P.AQN:  .BYTE   1, 8, 2, 0                          ;
                                                  00000000  022FB          .LONG   0                                         ;
                            2E  45  53  4C  41  46  2E  07  022FF          .ASCII  <7>\.FALSE.\                              ;
                                                            02307          .BLKB   1                                         ;
                                                  00000002  02308          .LONG   2                                         ;
                                        00002274  00002265  0230C P.AQL:  .LONG   8805, 8820                                ;
                                                  00000000  02314          .LONG   0                                         ;
                                                            02318 P.AQO:  .BLKB   0                                         ;
                                                        02  02318 P.AQP:  .BYTE   2                                         ;
```

```
                    01  02319          .BYTE   1
                0007    0231A          .WORD   7
                    03  0231C          .BYTE   3
                    03  0231D          .BYTE   3
                0009    0231E          .WORD   9
                    09  02320          .BYTE   9
                    0F  02321          .BYTE   15
                0009    02322          .WORD   9
                    05  02324          .BYTE   5
                    04  02325          .BYTE   4
                000E    02326          .WORD   14
                    04  02328          .BYTE   4
                    07  02329          .BYTE   7
                0012    0232A          .WORD   18
                    01  0232C          .BYTE   1
                    0C  0232D          .BYTE   12
                0016    0232E          .WORD   22
            00000000    02330          .LONG   0
                    01  02334          .BYTE   1
                    02  02335          .BYTE   2
                0016    02336          .WORD   22
            00000000    02338          .LONG   0
                    03  0233C          .BYTE   3
                    0E  0233D          .BYTE   14
                0009    0233E          .WORD   9
                    09  02340          .BYTE   9
                    0F  02341          .BYTE   15
                0009    02342          .WORD   9
                    04  02344          .BYTE   4
                    12  02345          .BYTE   18
                0012    02346          .WORD   18
                    01  02348          .BYTE   1
                    17  02349          .BYTE   23
                0016    0234A          .WORD   22
            00000000    0234C          .LONG   0
                    05  02350          .BYTE   5
                    19  02351          .BYTE   25
                000E    02352          .WORD   14
                    04  02354          .BYTE   4
                    1C  02355          .BYTE   28
                0012    02356          .WORD   18
                    01  02358          .BYTE   1
                    21  02359          .BYTE   33
                0016    0235A          .WORD   22
            00000000    0235C          .LONG   0
                    05  02360          .BYTE   5
                    24  02361          .BYTE   36
                000E    02362          .WORD   14
                    04  02364          .BYTE   4
                    26  02365          .BYTE   38
                0012    02366          .WORD   18
                    01  02368          .BYTE   1
                    2A  02369          .BYTE   42
                0016    0236A          .WORD   22
            00000000    0236C          .LONG   0
            00000000    02370          .LONG   0
00002259 00002295 00000C8D 000020D9 0000200D 00002001  02374 P.AQQ:  .LONG   8193, 8205, 8409, 3213, 8853, 8793, 8841, -
```

```
00000000 00000000 00000001 00000000 00002295 00002289 0238C                    8853, 0, 1, 0, 0, 0
                                             00000000 023A4
                                             00000005 023A8                .LONG   5
         3E040000 3C008800 2E01883E 24000007 5F000007 023AC P.AQR:  .LONG   1593835527, 603979783, 771852350, -
                                                                            1006667776, 1040449536
                                                    00 03 023C0 P.AQT:  .BYTE   3, 0
                                                      000D 023C2            .WORD   13
                                  00 00 00 00 00 00 32 32 023C4            .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                               4C 51 45 03 023CC            .ASCII  <3>\EQL\
                                                    00 03 023D0 P.AQU:  .BYTE   3, 0
                                                      000D 023D2            .WORD   13
                                  00 00 00 00 00 00 32 32 023D4            .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                               55 4C 51 45 04 023DC            .ASCII  <4>\EQLU\
                                                    00 03 023E1 P.AQV:  .BYTE   3, 0
                                                      000E 023E3            .WORD   14
                                  00 00 00 00 00 00 32 32 023E5            .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                               51 45 4E 03 023ED            .ASCII  <3>\NEQ\
                                                    00 03 023F1 P.AQW:  .BYTE   3, 0
                                                      000E 023F3            .WORD   14
                                  00 00 00 00 00 00 32 32 023F5            .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                               55 51 45 4E 04 023FD            .ASCII  <4>\NEQU\
                                                    00 03 02402 P.AQX:  .BYTE   3, 0
                                                      000F 02404            .WORD   15
                                  00 00 00 00 00 00 32 32 02406            .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                               52 54 47 03 0240E            .ASCII  <3>\GTR\
                                                    00 03 02412 P.AQY:  .BYTE   3, 0
                                                      0010 02414            .WORD   16
                                  00 00 00 00 00 00 32 32 02416            .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                               55 52 54 47 04 0241E            .ASCII  <4>\GTRU\
                                                    00 03 02423 P.AQZ:  .BYTE   3, 0
                                                      0011 02425            .WORD   17
                              00 00 00 00 00 00 32 32 02427            .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                           51 45 47 03 0242F            .ASCII  <3>\GEQ\
                                                    00 03 02433 P.ARA:  .BYTE   3, 0
                                                      0012 02435            .WORD   18
                              00 00 00 00 00 00 32 32 02437            .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                           55 51 45 47 04 0243F            .ASCII  <4>\GEQU\
                                                    00 03 02444 P.ARB:  .BYTE   3, 0
                                                      0013 02446            .WORD   19
                              00 00 00 00 00 00 32 32 02448            .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                           53 53 4C 03 02450            .ASCII  <3>\LSS\
                                                    00 03 02454 P.ARC:  .BYTE   3, 0
                                                      0014 02456            .WORD   20
                              00 00 00 00 00 00 32 32 02458            .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                           55 53 53 4C 04 02460            .ASCII  <4>\LSSU\
                                                    00 03 02465 P.ARD:  .BYTE   3, 0
                                                      0015 02467            .WORD   21
                              00 00 00 00 00 00 32 32 02469            .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                           51 45 4C 03 02471            .ASCII  <3>\LEQ\
                                                    00 03 02475 P.ARE:  .BYTE   3, 0
                                                      0016 02477            .WORD   22
                              00 00 00 00 00 00 32 32 02479            .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                           55 51 45 4C 04 02481            .ASCII  <4>\LEQU\
                                                    00 02 02486 P.ARF:  .BYTE   2, 0
                                                      001E 02488            .WORD   30
                              00 00 00 00 00 00 C8 28 0248A            .BYTE   40, -56, 0, 0, 0, 0, 0, 0
                                           54 4F 4E 03 02492            .ASCII  <3>\NOT\
```

```
                                                00 03  02496 P.ARG:   .BYTE   3, 0
                                                  001F  02498          .WORD   31
                        00 00 00 00 00 00 1E 1E 0249A          .BYTE   30, 30, 0, 0, 0, 0, 0, 0
                                          44 4E 41 03  024A2          .ASCII  <3>\AND\
                                                00 03  024A6 P.ARH:   .BYTE   3, 0
                                                  0020  024A8          .WORD   32
                        00 00 00 00 00 00 14 14 024AA          .BYTE   20, 20, 0, 0, 0, 0, 0, 0
                                          52 4F 02  024B2          .ASCII  <2>\OR\
                                                00 03  024B5 P.ARI:   .BYTE   3, 0
                                                  0022  024B7          .WORD   34
                        00 00 00 00 00 00 0A 0A 024B9          .BYTE   10, 10, 0, 0, 0, 0, 0, 0
                                          56 51 45 03  024C1          .ASCII  <3>\EQV\
                                                00 03  024C5 P.ARJ:   .BYTE   3, 0
                                                  0021  024C7          .WORD   33
                        00 00 00 00 00 00 0A 0A 024C9          .BYTE   10, 10, 0, 0, 0, 0, 0, 0
                                          52 4F 58 03  024D1          .ASCII  <3>\XOR\
                                                00 03  024D5 P.ARK:   .BYTE   3, 0
                                                  0025  024D7          .WORD   37
                        00 00 00 00 00 00 46 46 024D9          .BYTE   70, 70, 0, 0, 0, 0, 0, 0
                                          44 4F 4D 03  024E1          .ASCII  <3>\MOD\
                                                       024E5          .BLKB   3
                                            00000012   024E8          .LONG   18
0000238F 0000237F 0000236E 0000235E 0000234D 0000233D  024EC P.AQS:   .LONG   9021, 9037, 9054, 9070, 9087, 9103, 9120, -
000023F2 000023E2 000023D1 000023C1 000023B0 000023A0  02504                  9136, 9153, 9169, 9186, 9202, 9219, 9235, -
00002452 00002442 00002432 00002423 00002413 00002403  0251C                  9251, 9266, 9282, 9298
                                                01 02  02534 P.ARM:   .BYTE   2, 1
                                                  0002  02536          .WORD   2
                        00 00 00 00 00 00 00 00 02538          .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                5C 01  02540          .ASCII  <1><92>
                                                01 03  02542 P.ARN:   .BYTE   3, 1
                                                  0003  02544          .WORD   3
                        00 00 00 00 00 00 00 00 02546          .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                5C 01  0254E          .ASCII  <1><92>
                                                02 02  02550 P.ARO:   .BYTE   2, 2
                                                  000B  02552          .WORD   11
                        00 00 00 00 00 00 C8 05 02554          .BYTE   5, -56, 0, 0, 0, 0, 0, 0
                                                28 01  0255C          .ASCII  <1>\(\
                                                02 04  0255E P.ARP:   .BYTE   4, 2
                                                  000C  02560          .WORD   12
                        00 00 00 00 00 00 06 C8 02562          .BYTE   -56, 6, 0, 0, 0, 0, 0, 0
                                                29 01  0256A          .ASCII  <1>\)\
                                                02 04  0256C P.ARQ:   .BYTE   4, 2
                                                  0031  0256E          .WORD   49
                        00 00 00 00 00 00 6E C8 02570          .BYTE   -56, 110, 0, 0, 0, 0, 0, 0
                                                3C 01  02578          .ASCII  <1>\<\
                                                00 02  0257A P.ARR:   .BYTE   2, 0
                                                  0003  0257C          .WORD   3
                        00 00 00 00 00 00 C8 64 0257E          .BYTE   100, -56, 0, 0, 0, 0, 0, 0
                                                2E 01  02586          .ASCII  <1>\.\
                                                00 02  02588 P.ARS:   .BYTE   2, 0
                                                  0003  0258A          .WORD   3
                        00 00 00 00 00 00 C8 64 0258C          .BYTE   100, -56, 0, 0, 0, 0, 0, 0
                                                40 01  02594          .ASCII  <1>\@\
                                                00 02  02596 P.ART:   .BYTE   2, 0
                                                  0004  02598          .WORD   4
                        00 00 00 00 00 00 C8 5A 0259A          .BYTE   90, -56, 0, 0, 0, 0, 0, 0
                                                2B 01  025A2          .ASCII  <1>\+\
```

```
                                                00  02  02594 P.ARU:   .BYTE   2, 0
                                                  0005  02596         .WORD   5
                                00  00  00  00  00  00  C8  5A  025A8         .BYTE   90, -56, 0, 0, 0, 0, 0, 0
                                                2D  01  025B0         .ASCII  <1>\-\
                                                00  03  025B2 P.ARV:   .BYTE   3, 0
                                                  0026  025B4         .WORD   38
                                00  00  00  00  00  00  50  50  025B6         .BYTE   80, 80, 0, 0, 0, 0, 0, 0
                                                40  01  025BE         .ASCII  <1>\@\
                                                00  03  025C0 P.ARW:   .BYTE   3, 0
                                                  0008  025C2         .WORD   8
                                00  00  00  00  00  00  46  46  025C4         .BYTE   70, 70, 0, 0, 0, 0, 0, 0
                                                2A  01  025CC         .ASCII  <1>\*\
                                                00  03  025CE P.ARX:   .BYTE   3, 0
                                                  0009  025D0         .WORD   9
                                00  00  00  00  00  00  46  46  025D2         .BYTE   70, 70, 0, 0, 0, 0, 0, 0
                                                2F  01  025DA         .ASCII  <1>\/\
                                                00  03  025DC P.ARY:   .BYTE   3, 0
                                                  0006  025DE         .WORD   6
                                00  00  00  00  00  00  3C  3C  025E0         .BYTE   60, 60, 0, 0, 0, 0, 0, 0
                                                2B  01  025E8         .ASCII  <1>\+\
                                                00  03  025EA P.ARZ:   .BYTE   3, 0
                                                  0007  025EC         .WORD   7
                                00  00  00  00  00  00  3C  3C  025EE         .BYTE   60, 60, 0, 0, 0, 0, 0, 0
                                                2D  01  025F6         .ASCII  <1>\-\
                                                0000000E  025F8         .LONG   14
000024F7  000024E9  000024DB  000024CD  000024BF  000024B1  025FC P.ARL:   .LONG   9393, 9407, 9421, 9435, 9449, 9463, 9477, -
0000254B  0000253D  0000252F  00002521  00002513  00002505  02614                  9491, 9505, 9519, 9533, 9547, 9561, 9575
                                        00002567  00002559  0262C
                                                00000000  02634         .LONG   0
                                                          02638 P.ASA:  .BLKB   0
                                                00000000  02638         .LONG   0
                                                          0263C P.ASB:  .BLKB   0
                                                00000000  0263C         .LONG   0
                                                          02640 P.ASC:  .BLKB   0
                                                    02    02640 P.ASD:  .BYTE   2
                                                    01    02641         .BYTE   1
                                                  0005    02642         .WORD   5
                                                    03    02644         .BYTE   3
                                                    03    02645         .BYTE   3
                                                  0007    02646         .WORD   7
                                                    09    02648         .BYTE   9
                                                    0F    02649         .BYTE   15
                                                  0007    0264A         .WORD   7
                                                    01    0264C         .BYTE   1
                                                    0C    0264D         .BYTE   12
                                                  000B    0264E         .WORD   11
                                                00000000  02650         .LONG   0
                                                    01    02654         .BYTE   1
                                                    02    02655         .BYTE   2
                                                  000B    02656         .WORD   11
                                                00000000  02658         .LONG   0
                                                    03    0265C         .BYTE   3
                                                    0E    0265D         .BYTE   14
                                                  0007    0265E         .WORD   7
                                                    09    02660         .BYTE   9
                                                    0F    02661         .BYTE   15
                                                  0007    02662         .WORD   7
```

```
                                                    01  02664          .BYTE   1
                                                    17  02665          .BYTE   23
                                                  000B  02666          .WORD   11
                                              00000000  02668          .LONG   0
                                              00000000  0266C          .LONG   0
000025B5  000025BD  00000C8D  00002579  00002469  00002329  02670 P.ASE:  .LONG   9001, 9321, 9593, 3213, 9661, 9653, 9657, -
00000000  00000000  00000001  00000000  000025BD  000025B9  02688                 9661, 0, 1, 0, 0, 0
                                              00000000  026A0
                                              00000007  026A4          .LONG   7
5B000800  2E058838  5E010800  3E042800  3D042800  3C00A800  026A8 P.ASF:  .LONG   1006675968, 1023682560, 1040459776, -
                                              5D040000  026C0                 1577125888, 772114488, 1526728704, -
                                                                              1560543232
                                                 00  03  026C4 P.ASH:  .BYTE   3, 0
                                                   0030  026C6          .WORD   48
                          00  00  00  00  00  00  46  46  026C8          .BYTE   70, 70, 0, 0, 0, 0, 0, 0
                                              56  49  44  03  026D0          .ASCII  <3>\DIV\
                                                 00  03  026D4 P.ASI:  .BYTE   3, 0
                                                   0024  026D6          .WORD   36
                          00  00  00  00  00  00  46  46  026D8          .BYTE   70, 70, 0, 0, 0, 0, 0, 0
                                              44  4F  4D  03  026E0          .ASCII  <3>\MOD\
                                                 00  03  026E4 P.ASJ:  .BYTE   3, 0
                                                   0025  026E6          .WORD   37
                          00  00  00  00  00  00  46  46  026E8          .BYTE   70, 70, 0, 0, 0, 0, 0, 0
                                              4D  45  52  03  026F0          .ASCII  <3>\REM\
                                                 00  03  026F4 P.ASK:  .BYTE   3, 0
                                                   0018  026F6          .WORD   24
                          00  00  00  00  00  00  46  46  026F8          .BYTE   70, 70, 0, 0, 0, 0, 0, 0
                                              44  4E  41  03  02700          .ASCII  <3>\AND\
                                                 00  03  02704 P.ASL:  .BYTE   3, 0
                                                   0019  02706          .WORD   25
                          00  00  00  00  00  00  3C  3C  02708          .BYTE   60, 60, 0, 0, 0, 0, 0, 0
                                              52  4F  02  02710          .ASCII  <2>\OR\
                                                 00  02  02713 P.ASM:  .BYTE   2, 0
                                                   0017  02715          .WORD   23
                          00  00  00  00  00  00  C8  5A  02717          .BYTE   90, -56, 0, 0, 0, 0, 0, 0
                                              54  4F  4E  03  0271F          .ASCII  <3>\NOT\
                                                 00  03  02723 P.ASN:  .BYTE   3, 0
                                                   002A  02725          .WORD   42
                          00  00  00  00  00  00  32  32  02727          .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                              4E  49  02  0272F          .ASCII  <2>\IN\
                                                          02732          .BLKB   2
                                              00000007  02734          .LONG   7
00002690  00002681  00002671  00002661  00002651  00002641  02738 P.ASG:  .LONG   9793, 9809, 9825, 9841, 9857, 9872, 9888
                                              000026A0  02750
                                                 01  02  02754 P.ASP:  .BYTE   2, 1
                                                   0002  02756          .WORD   2
                          00  00  00  00  00  00  00  00  02758          .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                 5C  01  02760          .ASCII  <1><92>
                                                 01  03  02762 P.ASQ:  .BYTE   3, 1
                                                   0003  02764          .WORD   3
                          00  00  00  00  00  00  00  00  02766          .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                 5C  01  0276E          .ASCII  <1><92>
                                                 01  04  02770 P.ASR:  .BYTE   4, 1
                                                   0004  02772          .WORD   4
                          00  00  00  00  00  00  00  00  02774          .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                 5B  01  0277C          .ASCII  <1>\[\
                                                 01  03  0277E P.ASS:  .BYTE   3, 1
```

```
                        0005  02780                .WORD   5
00  00  00  00  00  00  00  00  02782                .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                        2E  01  0278A                .ASCII  <1>\.\
                        01  04  0278C P.AST:         .BYTE   4, 1
                          0007  0278E                .WORD   7
00  00  00  00  00  00  00  00  02790                .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                        5E  01  02798                .ASCII  <1>\^\
                        01  04  0279A P.ASU:         .BYTE   4, 1
                          000A  0279C                .WORD   10
00  00  00  00  00  00  00  00  0279E                .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                        28  01  027A6                .ASCII  <1>\(\
                        02  02  027A8 P.ASV:         .BYTE   2, 2
                          000B  027AA                .WORD   11
00  00  00  00  00  00  C8  05  027AC                .BYTE   5, -56, 0, 0, 0, 0, 0, 0
                        28  01  027B4                .ASCII  <1>\(\
                        02  04  027B6 P.ASW:         .BYTE   4, 2
                          000C  027B8                .WORD   12
00  00  00  00  00  00  06  C8  027BA                .BYTE   -56, 6, 0, 0, 0, 0, 0, 0
                        29  01  027C2                .ASCII  <1>\)\
                        02  02  027C4 P.ASX:         .BYTE   2, 2
                          0039  027C6                .WORD   57
00  00  00  00  00  00  C8  05  027C8                .BYTE   5, -56, 0, 0, 0, 0, 0, 0
                        5B  01  027D0                .ASCII  <1>\[\
                        00  02  027D2 P.ASY:         .BYTE   2, 0
                          0004  027D4                .WORD   4
00  00  00  00  00  00  C8  3C  027D6                .BYTE   60, -56, 0, 0, 0, 0, 0, 0
                        2B  01  027DE                .ASCII  <1>\+\
                        00  02  027E0 P.ASZ:         .BYTE   2, 0
                          0005  027E2                .WORD   5
00  00  00  00  00  00  C8  3C  027E4                .BYTE   60, -56, 0, 0, 0, 0, 0, 0
                        2D  01  027EC                .ASCII  <1>\-\
                        00  03  027EE P.ATA:         .BYTE   3, 0
                          000A  027F0                .WORD   10
00  00  00  00  00  00  50  50  027F2                .BYTE   80, 80, 0, 0, 0, 0, 0, 0
                    2A  2A  02  027FA                .ASCII  <2>\**\
                        00  03  027FD P.ATB:         .BYTE   3, 0
                          0008  027FF                .WORD   8
00  00  00  00  00  00  46  46  02801                .BYTE   70, 70, 0, 0, 0, 0, 0, 0
                        2A  01  02809                .ASCII  <1>\*\
                        00  03  0280B P.ATC:         .BYTE   3, 0
                          0009  0280D                .WORD   9
00  00  00  00  00  00  46  46  0280F                .BYTE   70, 70, 0, 0, 0, 0, 0, 0
                        2F  01  02817                .ASCII  <1>\/\
                        00  03  02819 P.ATD:         .BYTE   3, 0
                          0006  0281B                .WORD   6
00  00  00  00  00  00  3C  3C  0281D                .BYTE   60, 60, 0, 0, 0, 0, 0, 0
                        2B  01  02825                .ASCII  <1>\+\
                        00  03  02827 P.ATE:         .BYTE   3, 0
                          0007  02829                .WORD   7
00  00  00  00  00  00  3C  3C  0282B                .BYTE   60, 60, 0, 0, 0, 0, 0, 0
                        2D  01  02833                .ASCII  <1>\-\
                        00  03  02835 P.ATF:         .BYTE   3, 0
                          0013  02837                .WORD   19
00  00  00  00  00  00  32  32  02839                .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                        3C  01  02841                .ASCII  <1>\<\
                        00  03  02843 P.ATG:         .BYTE   3, 0
                          0015  02845                .WORD   21
```

```
                                00  00  00  00  00  00  32  32  02847           .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                                    3D  3C  02  0284F           .ASCII  <2>\<=\
                                                        00  03  02852 P.ATH:    .BYTE   3, 0
                                                          000F  02854           .WORD   15
                                00  00  00  00  00  00  32  32  02856           .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                                        3E  01  0285E           .ASCII  <1>\>\
                                                        00  03  02860 P.ATI:    .BYTE   3, 0
                                                          0011  02862           .WORD   17
                                00  00  00  00  00  00  32  32  02864           .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                                    3D  3E  02  0286C           .ASCII  <2>\>=\
                                                        00  03  0286F P.ATJ:    .BYTE   3, 0
                                                          000D  02871           .WORD   13
                                00  00  00  00  00  00  32  32  02873           .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                                        3D  01  0287B           .ASCII  <1>\=\
                                                        00  03  0287D P.ATK:    .BYTE   3, 0
                                                          000E  0287F           .WORD   14
                                00  00  00  00  00  00  32  32  02881           .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                                    3E  3C  02  02889           .ASCII  <2>\<>\
                                                  00000016  0288C           .LONG   22
00002717  00002709  000026FB  000026ED  000026DF  000026D1  02890 P.ASO:    .LONG   9937, 9951, 9965, 9979, 9993, 10007, -
0000276B  0000275D  0000274F  00002741  00002733  00002725  028A8                   10021, 10035, 10049, 10063, 10077, 10091, -
000027C0  000027B2  000027A4  00002796  00002788  0000277A  028C0                   10106, 10120, 10134, 10148, 10162, 10176, -
                    000027FA  000027EC  000027DD  000027CF  028D8                   10191, 10205, 10220, 10234
                                                        00  00  028E8 P.ATM:    .BYTE   0, 0
                                                          0002  028EA           .WORD   2
                                        00  00  00  00  028EC           .BYTE   0, 0, 0, 0
                                                        5D  01  028F0           .ASCII  <1>\]\
                                                        02  00  028F2 P.ATN:    .BYTE   0, 2
                                                          0003  028F4           .WORD   3
                                        00  00  00  00  028F6           .BYTE   0, 0, 0, 0
                                                        3A  01  028FA           .ASCII  <1>\:\
                                                        00  00  028FC P.ATO:    .BYTE   0, 0
                                                          0001  028FE           .WORD   1
                                        00  00  00  00  02900           .BYTE   0, 0, 0, 0
                                                        2C  01  02904           .ASCII  <1>\,\
                                                                02906           .BLKB   2
                                                  00000003  02908           .LONG   3
00002879  0000286F  00002865  0290C P.ATL:    .LONG   10341, 10351, 10361
                                00  02  28  01  02918 P.ATQ:    .BYTE   1, 40, 2, 0
                                              00000001  0291C           .LONG   1
                            45  55  52  54  04  02920           .ASCII  <4>\TRUE\
                                00  02  28  01  02925 P.ATR:    .BYTE   1, 40, 2, 0
                                              00000000  02929           .LONG   0
                        45  53  4C  41  46  05  0292D           .ASCII  <5>\FALSE\
                                00  06  00  01  02933 P.ATS:    .BYTE   1, 0, 6, 0
                                              00000000  02937           .LONG   0
                                4C  49  4E  03  0293B           .ASCII  <3>\NIL\
                                                    0293F           .BLKB   1
                                              00000003  02940           .LONG   3
00002880  000028A2  00002895  02944 P.ATP:    .LONG   10389, 10402, 10416
                    00000000  02950           .LONG   0
                              02954 P.ATT:    .BLKB   0
                                02  02954 P.ATU:    .BYTE   2
                                01  02955           .BYTE   1
                              0009  02956           .WORD   9
                                03  02958           .BYTE   3
                                03  02959           .BYTE   3
```

```
                000B  0295A         .WORD   11
                  09  0295C         .BYTE   9
                  0F  0295D         .BYTE   15
                000B  0295E         .WORD   11
                  05  02960         .BYTE   5
                  04  02961         .BYTE   4
                0012  02962         .WORD   18
                  04  02964         .BYTE   4
                  07  02965         .BYTE   7
                0017  02966         .WORD   23
                  07  02968         .BYTE   7
                  0A  02969         .BYTE   10
                001C  0296A         .WORD   28
                  0A  0296C         .BYTE   10
                  0D  0296D         .BYTE   13
                0021  0296E         .WORD   33
                  01  02970         .BYTE   1
                  0C  02971         .BYTE   12
                0021  02972         .WORD   33
            00000000  02974         .LONG   0
                  01  02978         .BYTE   1
                  02  02979         .BYTE   2
                0021  0297A         .WORD   33
            00000000  0297C         .LONG   0
                  03  02980         .BYTE   3
                  0E  02981         .BYTE   14
                000B  02982         .WORD   11
                  09  02984         .BYTE   9
                  0F  02985         .BYTE   15
                000B  02986         .WORD   11
                  05  02988         .BYTE   5
                  10  02989         .BYTE   16
                0012  0298A         .WORD   18
                  04  0298C         .BYTE   4
                  12  0298D         .BYTE   18
                0017  0298E         .WORD   23
                  07  02990         .BYTE   7
                  15  02991         .BYTE   21
                001C  02992         .WORD   28
                  01  02994         .BYTE   1
                  17  02995         .BYTE   23
                0021  02996         .WORD   33
            00000000  02998         .LONG   0
                  05  0299C         .BYTE   5
                  19  0299D         .BYTE   25
                0012  0299E         .WORD   18
                  04  029A0         .BYTE   4
                  1C  029A1         .BYTE   28
                0017  029A2         .WORD   23
                  07  029A4         .BYTE   7
                  1F  029A5         .BYTE   31
                001C  029A6         .WORD   28
                  01  029A8         .BYTE   1
                  21  029A9         .BYTE   33
                0021  029AA         .WORD   33
            00000000  029AC         .LONG   0
                  05  029B0         .BYTE   5
```

```
                                                          24   029B1            .BYTE   36                                    ;
                                                        0012   029B2            .WORD   18                                    ;
                                                          04   029B4            .BYTE   4                                     ;
                                                          26   029B5            .BYTE   38                                    ;
                                                        0017   029B6            .WORD   23                                    ;
                                                          07   029B8            .BYTE   7                                     ;
                                                          28   029B9            .BYTE   40                                    ;
                                                        001C   029BA            .WORD   28                                    ;
                                                          01   029BC            .BYTE   1                                     ;
                                                          2A   029BD            .BYTE   42                                    ;
                                                        0021   029BE            .WORD   33                                    ;
                                                    00000000   029C0            .LONG   0                                     ;
                                                          05   029C4            .BYTE   5                                     ;
                                                          2C   029C5            .BYTE   44                                    ;
                                                        0012   029C6            .WORD   18                                    ;
                                                          04   029C8            .BYTE   4                                     ;
                                                          2D   029C9            .BYTE   45                                    ;
                                                        0017   029CA            .WORD   23                                    ;
                                                          07   029CC            .BYTE   7                                     ;
                                                          2E   029CD            .BYTE   46                                    ;
                                                        001C   029CE            .WORD   28                                    ;
                                                          01   029D0            .BYTE   1                                     ;
                                                          2F   029D1            .BYTE   47                                    ;
                                                        0021   029D2            .WORD   33                                    ;
                                                    00000000   029D4            .LONG   0                                     ;
                                                    00000000   029D8            .LONG   0                                     ;
00002889  000028D1  00000C8D  0000280D  000026B5    00002625   029DC P.ATV:     .LONG   9765, 9909, 10253, 3213, 10449, 10377, -   ;
00000000  00000000  00000001  00000001  000028D1    000028C1   029F4                    10433, 10449, 1, 1, 0, 0, 0           ;
                                                    00000000   02A0C                                                          ;
                                                    0000000B   02A10            .LONG   11                                    ;
21002800  7C002800  26000800  25000002  5F000007    24000007   02A14 P.ATW:     .LONG   603979783, 1593835527, 620756994, -   ;
          3E042800  3C00A800  2D088850  5E012800    3D042800   02A2C                    637536256, 2080385024, 553658368, -   ;
                                                                                        1023682560, 1577134080, 755533904, -  ;
                                                                                        1006675968, 1040459776                ;
                                                    00000000   02A40            .LONG   0                                     ;
                                                               02A44 P.ATX:     .BLKB   0
                                                       01  02  02A44 P.ATZ:     .BYTE   2, 1                                  ;
                                                        0002   02A46            .WORD   2                                     ;
                          00  00  00  00  00  00   00  00   02A48            .BYTE   0, 0, 0, 0, 0, 0, 0, 0                ;
                                                       5C  01  02A50            .ASCII  <1><92>                               ;
                                                       01  03  02A52 P.AUA:     .BYTE   3, 1                                  ;
                                                        0003   02A54            .WORD   3                                     ;
                          00  00  00  00  00  00   00  00   02A56            .BYTE   0, 0, 0, 0, 0, 0, 0, 0                ;
                                                       5C  01  02A5E            .ASCII  <1><92>                               ;
                                                       01  04  02A60 P.AUB:     .BYTE   4, 1                                  ;
                                                        0004   02A62            .WORD   4                                     ;
                          00  00  00  00  00  00   00  00   02A64            .BYTE   0, 0, 0, 0, 0, 0, 0, 0                ;
                                                       28  01  02A6C            .ASCII  <1>\(\                                ;
                                                       01  03  02A6E P.AUC:     .BYTE   3, 1                                  ;
                                                        0005   02A70            .WORD   5                                     ;
                          00  00  00  00  00  00   00  00   02A72            .BYTE   0, 0, 0, 0, 0, 0, 0, 0                ;
                                                       2E  01  02A7A            .ASCII  <1>\.\                                ;
                                                       02  02  02A7C P.AUD:     .BYTE   2, 2                                  ;
                                                        000B   02A7E            .WORD   11                                    ;
                          00  00  00  00  00  00   C8  05   02A80            .BYTE   5, -56, 0, 0, 0, 0, 0, 0              ;
                                                       28  01  02A88            .ASCII  <1>\(\                                ;
                                                       02  04  02A8A P.AUE:     .BYTE   4, 2                                  ;
```

```
                    000C  02A8C                .WORD   12
00  00  00  00  00  00  06  C8  02A8E          .BYTE   -56, 6, 0, 0, 0, 0, 0, 0
                    29  01  02A96               .ASCII  <1>\)\
                    00  02  02A98 P.AUF:       .BYTE   2, 0
                      0004  02A9A               .WORD   4
00  00  00  00  00  00  C8  46  02A9C          .BYTE   70, -56, 0, 0, 0, 0, 0, 0
                    2B  01  02AA4               .ASCII  <1>\+\
                    00  02  02AA6 P.AUG:       .BYTE   2, 0
                      0005  02AA8               .WORD   5
00  00  00  00  00  00  C8  46  02AAA          .BYTE   70, -56, 0, 0, 0, 0, 0, 0
                    2D  01  02AB2               .ASCII  <1>\-\
                    00  02  02AB4 P.AUH:       .BYTE   2, 0
                      001E  02AB6               .WORD   30
00  00  00  00  00  00  C8  1E  02AB8          .BYTE   30, -56, 0, 0, 0, 0, 0, 0
                    5E  01  02AC0               .ASCII  <1>\^\
                    00  03  02AC2 P.AUI:       .BYTE   3, 0
                      000A  02AC4               .WORD   10
00  00  00  00  00  00  5C  5A  02AC6          .BYTE   90, 92, 0, 0, 0, 0, 0, 0
                    2A  2A  02  02ACE           .ASCII  <2>\**\
                    00  03  02AD1 P.AUJ:       .BYTE   3, 0
                      0008  02AD3               .WORD   8
00  00  00  00  00  00  50  50  02AD5          .BYTE   80, 80, 0, 0, 0, 0, 0, 0
                    2A  01  02ADD               .ASCII  <1>\*\
                    00  03  02ADF P.AUK:       .BYTE   3, 0
                      0009  02AE1               .WORD   9
00  00  00  00  00  00  50  50  02AE3          .BYTE   80, 80, 0, 0, 0, 0, 0, 0
                    2F  01  02AEB               .ASCII  <1>\/\
                    00  03  02AED P.AUL:       .BYTE   3, 0
                      0006  02AEF               .WORD   6
00  00  00  00  00  00  3C  3C  02AF1          .BYTE   60, 60, 0, 0, 0, 0, 0, 0
                    2B  01  02AF9               .ASCII  <1>\+\
                    00  03  02AFB P.AUM:       .BYTE   3, 0
                      0007  02AFD               .WORD   7
00  00  00  00  00  00  3C  3C  02AFF          .BYTE   60, 60, 0, 0, 0, 0, 0, 0
                    2D  01  02B07               .ASCII  <1>\-\
                    00  03  02B09 P.AUN:       .BYTE   3, 0
                      0023  02B0B               .WORD   35
00  00  00  00  00  00  37  37  02B0D          .BYTE   55, 55, 0, 0, 0, 0, 0, 0
                    7C  7C  02  02B15           .ASCII  <2>\!!\
                    00  03  02B18 P.AUO:       .BYTE   3, 0
                      000F  02B1A               .WORD   15
00  00  00  00  00  00  32  32  02B1C          .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                    3E  01  02B24               .ASCII  <1>\>\
                    00  03  02B26 P.AUP:       .BYTE   3, 0
                      0013  02B28               .WORD   19
00  00  00  00  00  00  32  32  02B2A          .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                    3C  01  02B32               .ASCII  <1>\<\
                    00  03  02B34 P.AUQ:       .BYTE   3, 0
                      0015  02B36               .WORD   21
00  00  00  00  00  00  32  32  02B38          .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                    3E  5E  02  02B40           .ASCII  <2>\^>\
                    00  03  02B43 P.AUR:       .BYTE   3, 0
                      0011  02B45               .WORD   17
00  00  00  00  00  00  32  32  02B47          .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                    3C  5E  02  02B4F           .ASCII  <2>\^<\
                    00  03  02B52 P.AUS:       .BYTE   3, 0
                      000D  02B54               .WORD   13
```

```
                                00  00  00  00  00  00  32  32  02B56            .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                                    3D  01  02B5E            .ASCII  <1>\=\
                                                    00  03  02B60 P.AUT:     .BYTE   3, 0
                                                      000E  02B62            .WORD   14
                                00  00  00  00  00  00  32  32  02B64            .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                                    3D  5E  02  02B6C            .ASCII  <2>\^=\
                                                    00  03  02B6F P.AUU:     .BYTE   3, 0
                                                      0015  02B71            .WORD   21
                                00  00  00  00  00  00  32  32  02B73            .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                                    3D  3C  02  02B7B            .ASCII  <2>\<=\
                                                    00  03  02B7E P.AUV:     .BYTE   3, 0
                                                      0011  02B80            .WORD   17
                                00  00  00  00  00  00  32  32  02B82            .BYTE   50, 50, 0, 0, 0, 0, 0, 0
                                                    3D  3E  02  02B8A            .ASCII  <2>\>=\
                                                    00  03  02B8D P.AUW:     .BYTE   3, 0
                                                      001F  02B8F            .WORD   31
                                00  00  00  00  00  00  2D  2D  02B91            .BYTE   45, 45, 0, 0, 0, 0, 0, 0
                                                    26  01  02B99            .ASCII  <1>\&\
                                                    00  03  02B9B P.AUX:     .BYTE   3, 0
                                                      0020  02B9D            .WORD   32
                                00  00  00  00  00  00  28  28  02B9F            .BYTE   40, 40, 0, 0, 0, 0, 0, 0
                                                    7C  01  02BA7            .ASCII  <1>\|\
                                                            02BA9            .BLKB   3
                                                00000019    02BAC            .LONG   25
00002A07  000029F9  000029EB  000029DD  000029CF  000029C1  02BB0 P.ATY:     .LONG   10689, 10703, 10717, 10731, 10745, 10759, -
00002A5C  00002A4E  00002A3F  00002A31  00002A23  00002A15  02BC8                    10773, 10787, 10801, 10815, 10830, 10844, -
00002AB1  00002AA3  00002A95  00002A86  00002A78  00002A6A  02BE0                    10858, 10872, 10886, 10901, 10915, 10929, -
00002B0A  00002AFB  00002AEC  00002ADD  00002ACF  00002AC0  02BF8                    10944, 10959, 10973, 10988, 11003, 11018, -
                                                00002B18    02C10                    11032
                                                      01  03  02C14 P.AUY:     .BYTE   3, 1
                                                      0008  02C16            .WORD   8
                                00  00  00  00  00  00  00  00  02C18            .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                    3E  2D  02  02C20            .ASCII  <2>\->\
                                                    00  00  02C23 P.AVA:     .BYTE   0, 0
                                                      0002  02C25            .WORD   2
                                            00  00  00  00  02C27            .BYTE   0, 0, 0, 0
                                                    29  01  02C2B            .ASCII  <1>\)\
                                                    02  00  02C2D P.AVB:     .BYTE   0, 2
                                                      0003  02C2F            .WORD   3
                                            00  00  00  00  02C31            .BYTE   0, 0, 0, 0
                                                    3A  01  02C35            .ASCII  <1>\:\
                                                    00  00  02C37 P.AVC:     .BYTE   0, 0
                                                      0001  02C39            .WORD   1
                                            00  00  00  00  02C3B            .BYTE   0, 0, 0, 0
                                                    2C  01  02C3F            .ASCII  <1>\,\
                                                            02C41            .BLKB   3
                                                00000003    02C44            .LONG   3
                            00002BB4  00002BAA  00002BA0  02C48 P.AUZ:     .LONG   11168, 11178, 11188
                                                00000000    02C54            .LONG   0
                                                            02C58 P.AVD:     .BLKB   0
                                                00000000    02C58            .LONG   0
                                                            02C5C P.AVE:     .BLKB   0
                                                      01    02C5C P.AVF:     .BYTE   1
                                                      0E    02C5D            .BYTE   14
                                                    0005    02C5E            .WORD   5
                                                      03    02C60            .BYTE   3
                                                      01    02C61            .BYTE   1
```

```
0003  02C62          .WORD   3
  00  02C64          .BYTE   0
  0A  02C65          .BYTE   10
0021  02C66          .WORD   33
  01  02C68          .BYTE   1
  0F  02C69          .BYTE   15
0012  02C6A          .WORD   18
  00  02C6C          .BYTE   0
  0A  02C6D          .BYTE   10
0021  02C6E          .WORD   33
  01  02C70          .BYTE   1
  0E  02C71          .BYTE   14
0005  02C72          .WORD   5
  03  02C74          .BYTE   3
  01  02C75          .BYTE   1
0012  02C76          .WORD   18
  02  02C78          .BYTE   2
  01  02C79          .BYTE   1
000C  02C7A          .WORD   12
  06  02C7C          .BYTE   6
  01  02C7D          .BYTE   1
000C  02C7E          .WORD   12
  07  02C80          .BYTE   7
  01  02C81          .BYTE   1
000C  02C82          .WORD   12
  08  02C84          .BYTE   8
  01  02C85          .BYTE   1
000C  02C86          .WORD   12
  00  02C88          .BYTE   0
  10  02C89          .BYTE   16
0021  02C8A          .WORD   33
  01  02C8C          .BYTE   1
  01  02C8D          .BYTE   1
000C  02C8E          .WORD   12
  02  02C90          .BYTE   2
  01  02C91          .BYTE   1
000C  02C92          .WORD   12
  06  02C94          .BYTE   6
  01  02C95          .BYTE   1
000C  02C96          .WORD   12
  07  02C98          .BYTE   7
  01  02C99          .BYTE   1
000C  02C9A          .WORD   12
  08  02C9C          .BYTE   8
  01  02C9D          .BYTE   1
000C  02C9E          .WORD   12
  00  02CA0          .BYTE   0
  09  02CA1          .BYTE   9
0021  02CA2          .WORD   33
  01  02CA4          .BYTE   1
  0F  02CA5          .BYTE   15
0012  02CA6          .WORD   18
  03  02CA8          .BYTE   3
  08  02CA9          .BYTE   8
0021  02CAA          .WORD   33
  08  02CAC          .BYTE   8
  05  02CAD          .BYTE   5
```

```
0019  02CAE          .WORD   25
  07  02CB0          .BYTE   7
  06  02CB1          .BYTE   6
0019  02CB2          .WORD   25
  0D  02CB4          .BYTE   13
  12  02CB5          .BYTE   18
0019  02CB6          .WORD   25
  09  02CB8          .BYTE   9
  07  02CB9          .BYTE   7
0019  02CBA          .WORD   25
  00  02CBC          .BYTE   0
  10  02CBD          .BYTE   16
0021  02CBE          .WORD   33
  01  02CC0          .BYTE   1
  01  02CC1          .BYTE   1
001F  02CC2          .WORD   31
  04  02CC4          .BYTE   4
  01  02CC5          .BYTE   1
001D  02CC6          .WORD   29
  05  02CC8          .BYTE   5
  01  02CC9          .BYTE   1
001D  02CCA          .WORD   29
  00  02CCC          .BYTE   0
  08  02CCD          .BYTE   8
0021  02CCE          .WORD   33
  01  02CD0          .BYTE   1
  01  02CD1          .BYTE   1
001F  02CD2          .WORD   31
  00  02CD4          .BYTE   0
  09  02CD5          .BYTE   9
0021  02CD6          .WORD   33
  01  02CD8          .BYTE   1
  01  02CD9          .BYTE   1
001F  02CDA          .WORD   31
  00  02CDC          .BYTE   0
  09  02CDD          .BYTE   9
0021  02CDE          .WORD   33
  00  02CE0          .BYTE   0
  0B  02CE1          .BYTE   11
0021  02CE2          .WORD   33
  02  02CE4 P.AVG:   .BYTE   2
  01  02CE5          .BYTE   1
0008  02CE6          .WORD   8
  03  02CE8          .BYTE   3
  03  02CE9          .BYTE   3
000A  02CEA          .WORD   10
  09  02CEC          .BYTE   9
  0F  02CED          .BYTE   15
000A  02CEE          .WORD   10
  05  02CF0          .BYTE   5
  05  02CF1          .BYTE   5
0011  02CF2          .WORD   17
  04  02CF4          .BYTE   4
  09  02CF5          .BYTE   9
0016  02CF6          .WORD   22
  08  02CF8          .BYTE   8
  0B  02CF9          .BYTE   11
```

```
                    0000  02CFA          .WORD   0
                      01  02CFC          .BYTE   1
                      0C  02CFD          .BYTE   12
                    001B  02CFE          .WORD   27
                00000000  02D00          .LONG   0
                      01  02D04          .BYTE   1
                      02  02D05          .BYTE   2
                    001B  02D06          .WORD   27
                00000000  02D08          .LONG   0
                      03  02D0C          .BYTE   3
                      0E  02D0D          .BYTE   14
                    000A  02D0E          .WORD   10
                      09  02D10          .BYTE   9
                      0F  02D11          .BYTE   15
                    000A  02D12          .WORD   10
                      05  02D14          .BYTE   5
                      11  02D15          .BYTE   17
                    0011  02D16          .WORD   17
                      04  02D18          .BYTE   4
                      13  02D19          .BYTE   19
                    0016  02D1A          .WORD   22
                      08  02D1C          .BYTE   8
                      16  02D1D          .BYTE   22
                    0000  02D1E          .WORD   0
                      01  02D20          .BYTE   1
                      17  02D21          .BYTE   23
                    001B  02D22          .WORD   27
                00000000  02D24          .LONG   0
                      05  02D28          .BYTE   5
                      1A  02D29          .BYTE   26
                    0011  02D2A          .WORD   17
                      04  02D2C          .BYTE   4
                      1D  02D2D          .BYTE   29
                    0016  02D2E          .WORD   22
                      08  02D30          .BYTE   8
                      20  02D31          .BYTE   32
                    0000  02D32          .WORD   0
                      01  02D34          .BYTE   1
                      22  02D35          .BYTE   34
                    001B  02D36          .WORD   27
                00000000  02D38          .LONG   0
                      05  02D3C          .BYTE   5
                      25  02D3D          .BYTE   37
                    0011  02D3E          .WORD   17
                      04  02D40          .BYTE   4
                      27  02D41          .BYTE   39
                    0016  02D42          .WORD   22
                      08  02D44          .BYTE   8
                      29  02D45          .BYTE   41
                    0000  02D46          .WORD   0
                      01  02D48          .BYTE   1
                      2B  02D49          .BYTE   43
                    001B  02D4A          .WORD   27
                00000000  02D4C          .LONG   0
                00000000  02D50          .LONG   0
00002BC5  00002C61  00002BD9  00002B2D  000029C1  00002991  02D54 P.AVH:  .LONG   10641, 10689, 11053, 11225, 11361, 11205, -
00000001  00000000  00000001  00000000  00002BD9  00002BD5  02D6C          .LONG   11221, 11225, 0, 1, 0, 1, 0
```

```
                                                00000000  02D84
                                                00000007  02D88                   .LONG   7
5F000007  23000007  3D042800  3E042800  3C04A800  2A000000  02D8C P.AVI:    .LONG   704643072, 1006938112, 1040459776, -
                                                24000007  02DA4                           1023682560, 587202567, 1593835527, -
                                                                                          603979783
                                                      00  02  02DA8 P.AVK:    .BYTE   2, 0
                                                        0017  02DAA                   .WORD   23
                        00  00  00  00  00  00  C8  0B  02DAC                   .BYTE   11, -56, 0, 0, 0, 0, 0, 0
                                            54  4F  4E  03  02DB4                   .ASCII  <3>\NOT\
                                                      00  03  02DB8 P.AVL:    .BYTE   3, 0
                                                        0018  02DBA                   .WORD   24
                        00  00  00  00  00  00  0A  0A  02DBC                   .BYTE   10, 10, 0, 0, 0, 0, 0, 0
                                            44  4E  41  03  02DC4                   .ASCII  <3>\AND\
                                                      00  03  02DC8 P.AVM:    .BYTE   3, 0
                                                        0019  02DCA                   .WORD   25
                        00  00  00  00  00  00  0A  0A  02DCC                   .BYTE   10, 10, 0, 0, 0, 0, 0, 0
                                                52  4F  02  02DD4                   .ASCII  <2>\OR\
                                                      02  03  02DD7 P.AVN:    .BYTE   3, 2
                                                        002C  02DD9                   .WORD   44
                        00  00  00  00  00  00  0F  0F  02DDB                   .BYTE   15, 15, 0, 0, 0, 0, 0, 0
                                            54  4F  4E  03  02DE3                   .ASCII  <3>\NOT\
                                                            02DE7                   .BLKB   1
                                                00000004  02DE8                   .LONG   4
                    00002D54  00002D45  00002D35  00002D25  02DEC P.AVJ:    .LONG   11557, 11573, 11589, 11604
                                                      00  03  02DFC P.AVO:    .BYTE   3, 0
                                                        000E  02DFE                   .WORD   14
                        00  00  00  00  00  00  0F  0F  02E00                   .BYTE   15, 15, 0, 0, 0, 0, 0, 0
                                    3D  20  54  4F  4E  05  02E08                   .ASCII  <5>\NOT =\
                                                      00  03  02E0E P.AVP:    .BYTE   3, 0
                                                        0015  02E10                   .WORD   21
                        00  00  00  00  00  00  0F  0F  02E12                   .BYTE   15, 15, 0, 0, 0, 0, 0, 0
                                    3E  20  54  4F  4E  05  02E1A                   .ASCII  <5>\NOT >\
                                                      00  03  02E20 P.AVQ:    .BYTE   3, 0
                                                        0011  02E22                   .WORD   17
                00  00  00  00  00  00  00  0F  0F  02E24                   .BYTE   15, 15, 0, 0, 0, 0, 0, 0
                                    3C  20  54  4F  4E  05  02E2C                   .ASCII  <5>\NOT <\
                                                      01  02  02E32 P.AVS:    .BYTE   2, 1
                                                        0002  02E34                   .WORD   2
                00  00  00  00  00  00  00  00  02E36                   .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                5C  01  02E3E                   .ASCII  <1><92>
                                                      01  03  02E40 P.AVT:    .BYTE   3, 1
                                                        0003  02E42                   .WORD   3
                00  00  00  00  00  00  00  00  02E44                   .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                5C  01  02E4C                   .ASCII  <1><92>
                                                      01  04  02E4E P.AVU:    .BYTE   4, 1
                                                        0004  02E50                   .WORD   4
                00  00  00  00  00  00  00  00  02E52                   .BYTE   0, 0, 0, 0, 0, 0, 0, 0
                                                28  01  02E5A                   .ASCII  <1>\(\
                                                      02  02  02E5C P.AVV:    .BYTE   2, 2
                                                        003F  02E5E                   .WORD   63
                00  00  00  00  00  00  C8  0F  02E60                   .BYTE   15, -56, 0, 0, 0, 0, 0, 0
                                                3E  01  02E68                   .ASCII  <1>\>\
                                                      02  02  02E6A P.AVW:    .BYTE   2, 2
                                                        0040  02E6C                   .WORD   64
                00  00  00  00  00  00  C8  0F  02E6E                   .BYTE   15, -56, 0, 0, 0, 0, 0, 0
                                                3C  01  02E76                   .ASCII  <1>\<\
                                                      02  02  02E78 P.AVX:    .BYTE   2, 2
```

```
                                                          0041  02E7A          .WORD   65
                           00  00  00  00  00  00  C8   0F  02E7C          .BYTE   15, -56, 0, 0, 0, 0, 0, 0
                                                   3D   01  02E84          .ASCII  <1>\=\
                                                   02   02  02E86 P.AVY:   .BYTE   2, 2
                                                          000B  02E88          .WORD   11
                           00  00  00  00  00  00  C8   05  02E8A          .BYTE   5, -56, 0, 0, 0, 0, 0, 0
                                                   28   01  02E92          .ASCII  <1>\(\
                                                   02   04  02E94 P.AVZ:   .BYTE   4, 2
                                                          000C  02E96          .WORD   12
                           00  00  00  00  00  00  06   C8  02E98          .BYTE   -56, 6, 0, 0, 0, 0, 0, 0
                                                   29   01  02EA0          .ASCII  <1>\)\
                                                   00   03  02EA2 P.AWA:   .BYTE   3, 0
                                                          0009  02EA4          .WORD   9
                           00  00  00  00  00  00  1E   1E  02EA6          .BYTE   30, 30, 0, 0, 0, 0, 0, 0
                                                   2F   01  02EAE          .ASCII  <1>\/\
                                                   00   02  02EB0 P.AWB:   .BYTE   2, 0
                                                          0004  02EB2          .WORD   4
                           00  00  00  00  00  00  C8   19  02EB4          .BYTE   25, -56, 0, 0, 0, 0, 0, 0
                                                   2B   01  02EBC          .ASCII  <1>\+\
                                                   00   02  02EBE P.AWC:   .BYTE   2, 0
                                                          0005  02EC0          .WORD   5
                           00  00  00  00  00  00  C8   19  02EC2          .BYTE   25, -56, 0, 0, 0, 0, 0, 0
                                                   2D   01  02ECA          .ASCII  <1>\-\
                                                   00   03  02ECC P.AWD:   .BYTE   3, 0
                                                          0006  02ECE          .WORD   6
                           00  00  00  00  00  00  14   14  02ED0          .BYTE   20, 20, 0, 0, 0, 0, 0, 0
                                                   2B   01  02ED8          .ASCII  <1>\+\
                                                   00   03  02EDA P.AWE:   .BYTE   3, 0
                                                          0007  02EDC          .WORD   7
                           00  00  00  00  00  00  14   14  02EDE          .BYTE   20, 20, 0, 0, 0, 0, 0, 0
                                                   2D   01  02EE6          .ASCII  <1>\-\
                                                   00   03  02EE8 P.AWF:   .BYTE   3, 0
                                                          000F  02EEA          .WORD   15
                           00  00  00  00  00  00  0F   0F  02EEC          .BYTE   15, 15, 0, 0, 0, 0, 0, 0
                                                   3E   01  02EF4          .ASCII  <1>\>\
                                                   00   03  02EF6 P.AWG:   .BYTE   3, 0
                                                          0013  02EF8          .WORD   19
                           00  00  00  00  00  00  0F   0F  02EFA          .BYTE   15, 15, 0, 0, 0, 0, 0, 0
                                                   3C   01  02F02          .ASCII  <1>\<\
                                                   00   03  02F04 P.AWH:   .BYTE   3, 0
                                                          000D  02F06          .WORD   13
                           00  00  00  00  00  00  0F   0F  02F08          .BYTE   15, 15, 0, 0, 0, 0, 0, 0
                                                   3D   01  02F10          .ASCII  <1>\=\
                                                                02F12          .BLKB   2
                                                      00000010  02F14          .LONG   16
00002DF5  00002DE7  00002DD9  00002DCB  00002DBD  00002DAF  02F18 P.AVR:   .LONG   11695, 11709, 11723, 11737, 11751, 11765, -
00002E49  00002E3B  00002E2D  00002E1F  00002E11  00002E03  02F30                  11779, 11793, 11807, 11821, 11835, 11849, -
                    00002E81  00002E73  00002E65  00002E57  02F48                  11863, 11877, 11891, 11905
                                                   00   03  02F58 P.AWI:   .BYTE   3, 0
                                                          0008  02F5A          .WORD   8
                           00  00  00  00  00  00  1E   1E  02F5C          .BYTE   30, 30, 0, 0, 0, 0, 0, 0
                                                   2A   01  02F64          .ASCII  <1>\*\
                                                   00   00  02F66 P.AWK:   .BYTE   0, 0
                                                          0002  02F68          .WORD   2
                                           00  00  00   00  02F6A          .BYTE   0, 0, 0, 0
                                                   29   01  02F6E          .ASCII  <1>\)\
                                                   02   00  02F70 P.AWL:   .BYTE   0, 2
```

```
                                0003  02F72           .WORD   3
                          00  00  00  00  02F74           .BYTE   0, 0, 0, 0
                                3A  01  02F78           .ASCII  <1>\:\
                                00  00  02F7A P.AWM:    .BYTE   0, 0
                                  0001  02F7C           .WORD   1
                          00  00  00  00  02F7E           .BYTE   0, 0, 0, 0
                                2C  01  02F82           .ASCII  <1>\,\
                             00000003  02F84           .LONG   3
00002EF7  00002EED  00002EE3  02F88 P.AWJ:    .LONG   12003, 12013, 12023
                             00000000  02F94           .LONG   0
                                       02F98 P.AWN:    .BLKB   0
                             00000000  02F98           .LONG   0
                                       02F9C P.AWO:    .BLKB   0
                                   02  02F9C P.AWP:    .BYTE   2
                                   01  02F9D           .BYTE   1
                                 0006  02F9E           .WORD   6
                                   03  02FA0           .BYTE   3
                                   03  02FA1           .BYTE   3
                                 0008  02FA2           .WORD   8
                                   09  02FA4           .BYTE   9
                                   0F  02FA5           .BYTE   15
                                 0008  02FA6           .WORD   8
                                   04  02FA8           .BYTE   4
                                   07  02FA9           .BYTE   7
                                 000D  02FAA           .WORD   13
                                   01  02FAC           .BYTE   1
                                   0C  02FAD           .BYTE   12
                                 0010  02FAE           .WORD   16
                             00000000  02FB0           .LONG   0
                                   01  02FB4           .BYTE   1
                                   02  02FB5           .BYTE   2
                                 0010  02FB6           .WORD   16
                             00000000  02FB8           .LONG   0
                                   03  02FBC           .BYTE   3
                                   0E  02FBD           .BYTE   14
                                 0008  02FBE           .WORD   8
                                   09  02FC0           .BYTE   9
                                   0F  02FC1           .BYTE   15
                                 0008  02FC2           .WORD   8
                                   04  02FC4           .BYTE   4
                                   12  02FC5           .BYTE   18
                                 000D  02FC6           .WORD   13
                                   01  02FC8           .BYTE   1
                                   17  02FC9           .BYTE   23
                                 0010  02FCA           .WORD   16
                             00000000  02FCC           .LONG   0
                                   04  02FD0           .BYTE   4
                                   26  02FD1           .BYTE   38
                                 000D  02FD2           .WORD   13
                                   01  02FD4           .BYTE   1
                                   2A  02FD5           .BYTE   42
                                 0010  02FD6           .WORD   16
                             00000000  02FD8           .LONG   0
                             00000000  02FDC           .LONG   0
00002F05  00002F19  00002BD9  00002E95  00002D69  00002D09  02FE0 P.AWQ:    .LONG   11529, 11625, 11925, 11225, 12057, 12037, -
00000000  00000000  00000001  00000000  000028D1  00002F15  02FF8                   12053, 10449, 0, 1, 0, 0, 0
                                                  00000000  03010
```

```
00002CD1 0000153D 00001FC9 000018F9 000022F1 000025ED 03014 LANGUAGE_TABLE_PTRS:
         00000DA1 00001229 00002F5D 00001C45 00002959 0302C         .LONG   9709, 8945, 6393, 8137, 5437, 11473, -
                                                                    10585, 7237, 12125, 4649, 3489

                                                                .PSECT  DBG$OWN,NOEXE,  PIC,2

                                                    00000 ADDRESS_LENGTH:
                                                                .BLKB   4
                                                    00004 ADDRESS_TYPE:
                                                                .BLKB   4
                                                    00008 BIF_TABLE:
                                                                .BLKB   4
                                                    0000C CASING_SIGNIFICANT:
                                                                .BLKB   4
                                                    00010 CHARPTR:.BLKB   4
                                                    00014 CHARTBL:.BLKB   1024
                                                    00414 COMPONENTS_IN_PATHNAME:
                                                                .BLKB   4
                                                    00418 ENFORCE_RECORD:
                                                                .BLKB   4
                                                    0041C EXPRESSION_RADIX:
                                                                .BLKB   4
                                                    00420 IDENT_OPERATOR_TABLE:
                                                                .BLKB   4
                                                    00424 INCOMPLETE_QUAL:
                                                                .BLKB   4
                                                    00428 MULTIPLE_SUBSCR:
                                                                .BLKB   4
                                                    0042C OPCHAR_OPERATOR_TABLE:
                                                                .BLKB   4
                                                    00430 PRIMARY_TABLE:
                                                                .BLKB   4
                                         00000000   00434 SAVED_TOKEN:
                                                                .LONG   0                               ;
                                                    00438 STATE_TABLE:
                                                                .BLKB   4
                                                    0043C SUBSCRIPT_TERM_TBL:
                                                                .BLKB   4
                                                    00440 PRIDTBL:.BLKB   4
                                         00000000   00444 TERMINATOR_CODE:
                                                                .LONG   0                               ;
                                         00000000   00448 TERMINATOR_LENGTH:
                                                                .LONG   0                               ;
                                                    0044C VARSTACK1:
                                                                .BLKB   80
                                                    0049C VARSTACK2:
                                                                .BLKB   80
                                                    004EC VARSTACK3:
                                                                .BLKB   80
                                                    0053C VARSTK_INDEX:
                                                                .BLKB   4

                                                                .PSECT  DBG$GLOBAL,NOEXE,  PIC,2

                                                    00000 DBG$GL_CHARTBL::
                                                                .BLKB   4
```

```
DBG$GL_CONVERT_TOKEN==
                        P.AAA
DBG$GL_DEPOSIT_TOKEN==
                        P.AAB
DBG$GL_IDENTITY_TOKEN==
                        P.AAC
DBG$GL_NEG_CONST_TOKEN==
                        P.AAD
DBG$GL_POS_CONST_TOKEN==
                        P.AAE
DBG$GL_NEG_SIGN_TOKEN==
                        P.AAF
DBG$GL_POS_SIGN_TOKEN==
                        P.AAG
TABLEBASE=              P.AAH
PERCENT_TABLE=          P.AAI
INITIATOR_TOKEN=        P.ACQ
TERMINATOR_TOKEN=       P.ACR
PRIMARY_TERM_TOKEN=     P.ACS
RADIX_OP_DEC=           P.ACT
RADIX_OP_HEX=           P.ACU
RADIX_OP_OCT=           P.ACV
RADIX_OP_BIN=           P.ACW
CURLOC_TOKEN=           P.ACX
CURVAL_TOKEN=           P.ACY
PREVLOC_TOKEN=          P.ACZ
ADDR_EXPR_OPTBL=        P.ADA
EMPTY_TERM_TBL=         P.ADM
COMMA_TERM_TBL=         P.ADN
EQUAL_TERM_TBL=         P.ADP
DO_TERM_TBL=            P.ADR
THEN_TERM_TBL=          P.ADT
COMCOL_TERM_TBL=        P.ADV
CMWHDO_TERM_TBL=        P.ADY
OPEN_TERM_TBL=          P.AEC
COMPAREN_TERM_TBL=      P.AEE
TO_TERM_TBL=            P.AEH
BY_TERM_TBL=            P.AEJ
BIT_SELECT_TERM_TBL=    P.AEM
SET_CONSTANT_TERM_TBL=
                        P.AEP
UNKNOWN_CHARTBL=        P.AET
UNKNOWN_IDENT_OPTBL=    P.AEU
UNKNOWN_OPCHAR_OPTBL=
                        P.AFG
UNKNOWN_SUBSCR_TERM_TBL=
                        P.AGF
UNKNOWN_NUMBER_TABLE=
                        P.AGK
UNKNOWN_PRID_TABLE=     P.AGL
UNKNOWN_FUNCTION_TABLE=
                        P.AGM
UNKNOWN_PRIMARY_TABLE=
                        P.AGN
UNKNOWN_TABLES=         P.AGO
ADA_CHARTBL=            P.AGP
ADA_IDENT_OPTBL=        P.AGQ
```

```
ADA_OPCHAR_OPTBL=    P.AGY
ADA_TICK_TOKEN=      P.AHT
ADA_SUBSCR_TERM_TBL=P.AID
ADA_PRID_TABLE=      P.AII
ADA_FUNCTION_TABLE=  P.AIJ
ADA_NUMBER_TABLE=    P.AIK
ADA_PRIMARY_TABLE=   P.AIL
ADA_TABLES=          P.AIM
BASIC_CHARTBL=       P.AIN
BASIC_IDENT_OPTBL=   P.AIO
BASIC_OPCHAR_OPTBL=  P.AIV
BASIC_SUBSCR_TERM_TBL=
                     P.AJT
BASIC_PRID_TABLE=    P.AJX
BASIC_FUNCTION_TABLE=
                     P.AJY
BASIC_NUMBER_TABLE=  P.AGK
BASIC_PRIMARY_TABLE=P.AJZ
BASIC_TABLES=        P.AKA
BLISS_CHARTBL=       P.AKB
BLISS_IDENT_OPTBL=   P.AKC
BLISS_OPCHAR_OPTBL=  P.ALB
BLISS_SUBSCR_TERM_TBL=
                     P.ALQ
BLISS_PRID_TABLE=    P.ALU
BLISS_FUNCTION_TABLE=
                     P.ALV
BLISS_NUMBER_TABLE=  P.AGK
BLISS_PRIMARY_TABLE=P.ALW
BLISS_TABLES=        P.ALX
C_CHARTBL=           P.ALY
C_IDENT_OPTBL=       P.ALZ
C_OPCHAR_OPTBL=      P.AMB
C_ADDR_OF_TOKEN=     P.AMZ
C_BIT_AND_TOKEN=     P.ANA
C_AND_TOKEN=         P.ANB
C_ADD_TOKEN=         P.ANC
C_MINUS_TOKEN=       P.AND
C_SUB_TOKEN=         P.ANE
C_ARROW_TOKEN=       P.ANF
C_INDIRECT_TOKEN=    P.ANG
C_SUBSCR_TERM_TBL=   P.ANH
C_PRID_TABLE=        P.ANL
C_FUNCTION_TABLE=    P.ANM
C_NUMBER_TABLE=      P.AGK
C_PRIMARY_TABLE=     P.ANN
C_TABLES=            P.ANO
COBOL_CHARTBL=       P.ANP
COBOL_IDENT_OPTBL=   P.ANQ
COBOL_NOT_EQL_TOKEN=P.ANX
COBOL_NOT_GTR_TOKEN=P.ANY
COBOL_NOT_LSS_TOKEN=P.ANZ
COBOL_OPCHAR_OPTBL=  P.AOA
COBOL_SUBSCR_TERM_TBL=
                     P.AOS
COBOL_PRID_TABLE=    P.AOW
COBOL_FUNCTION_TABLE=
```

```
                                P.AOX
COBOL_NUMBER_TABLE=             P.AOY
COBOL_PRIMARY_TABLE=            P.AOZ
COBOL_TABLES=                   P.APA
FORTRAN_CHARTBL=                P.APB
FORTRAN_IDENT_OPTBL=            P.APC
FORTRAN_OPCHAR_OPTBL=
                                P.APD
FORTRAN_INDIRECT_TOKEN=
                                P.APS
FORTRAN_DOT_TOKEN=              P.APT
FORTRAN_SPECIAL_OPTBL=
                                P.APU
FORTRAN_SUBSCR_TERM_TBL=
                                P.AQH
FORTRAN_PRID_TABLE=             P.AQL
FORTRAN_FUNCTION_TABLE=
                                P.AQO
FORTRAN_NUMBER_TABLE=
                                P.AGK
FORTRAN_PRIMARY_TABLE=
                                P.AQP
FORTRAN_TABLES=                 P.AQQ
MACRO_CHARTBL=                  P.AQR
MACRO_IDENT_OPTBL=              P.AQS
MACRO_OPCHAR_OPTBL=             P.ARL
MACRO_SUBSCR_TERM_TBL=
                                P.ASA
MACRO_PRID_TABLE=               P.ASB
MACRO_FUNCTION_TABLE=
                                P.ASC
MACRO_NUMBER_TABLE=             P.AGK
MACRO_PRIMARY_TABLE=            P.ASD
MACRO_TABLES=                   P.ASE
PASCAL_CHARTBL=                 P.ASF
PASCAL_IDENT_OPTBL=             P.ASG
PASCAL_OPCHAR_OPTBL=            P.ASO
PASCAL_SUBSCR_TERM_TBL=
                                P.ATL
PASCAL_PRID_TABLE=              P.ATP
PASCAL_FUNCTION_TABLE=
                                P.ATT
PASCAL_NUMBER_TABLE=            P.AGK
PASCAL_PRIMARY_TABLE=
                                P.ATU
PASCAL_TABLES=                  P.ATV
PLI_CHARTBL=                    P.ATW
PLI_IDENT_OPTBL=                P.ATX
PLI_OPCHAR_OPTBL=               P.ATY
PLI_ARROW_TOKEN=                P.AUY
PLI_SUBSCR_TERM_TBL=            P.AUZ
PLI_PRID_TABLE=                 P.AVD
PLI_FUNCTION_TABLE=             P.AVE
PLI_NUMBER_TABLE=               P.AVF
PLI_PRIMARY_TABLE=              P.AVG
PLI_TABLES=                     P.AVH
RPG_CHARTBL=                    P.AVI
```

```
RPG_IDENT_OPTBL=     P.AVJ
RPG_NOT_EQL_TOKEN=   P.AVO
RPG_NOT_GTR_TOKEN=   P.AVP
RPG_NOT_LSS_TOKEN=   P.AVQ
RPG_OPCHAR_OPTBL=    P.AVR
RPG_MULTIPLY_TOKEN=  P.AWI
RPG_SUBSCR_TERM_TBL=P.AWJ
RPG_PRID_TABLE=      P.AWN
RPG_FUNCTION_TABLE=  P.AWO
RPG_NUMBER_TABLE=    P.AVF
RPG_PRIMARY_TABLE=   P.AWP
RPG_TABLES=          P.AWQ
        .EXTRN  DBG$DATA_LENGTH
        .EXTRN  DBG$DEF_SYM_FIND
        .EXTRN  DBG$DUMP_HEX, DBG$ENUM_POS
        .EXTRN  DBG$ENUM_VAL, DBG$EVALOP_SET_LANGUAGE
        .EXTRN  DBG$EVAL_ADA_TICK
        .EXTRN  DBG$EVAL_ADDR_OPERATOR
        .EXTRN  DBG$EVAL_LANG_OPERATOR
        .EXTRN  DBG$GET_TEMPMEM
        .EXTRN  DBG$HASH_FIND, DBG$HASH_FIND_SETUP
        .EXTRN  DBG$PERFORM_TYPEID_CHECK
        .EXTRN  DBG$MAKE_SKELETON_DESC
        .EXTRN  DBG$MAP_DTYPE_CLASS
        .EXTRN  DBG$NCOB_PATHDESC_TO_CS
        .EXTRN  DBG$NCOPY_DESC, DBG$NEWLINE
        .EXTRN  DBG$NPATHDESC_TO_CS
        .EXTRN  DBG$NUM_BYTES, DBG$PRIM_TO_ADDR
        .EXTRN  DBG$PRIM_TO_VAL
        .EXTRN  DBG$PRINT, DBG$PRINT_SET_LANGUAGE
        .EXTRN  DBG$STA_GETSYMBOL
        .EXTRN  DBG$STA_SETCONTEXT
        .EXTRN  DBG$STA_SYMSIZE
        .EXTRN  DBG$STA_SYMTYPE
        .EXTRN  DBG$STA_SYMVALUE
        .EXTRN  DBG$STA_TYPEFCODE
        .EXTRN  DBG$STA_TYP_ARRAY
        .EXTRN  DBG$STA_TYP_ATOMIC
        .EXTRN  DBG$STA_TYP_DESCR
        .EXTRN  DBG$STA_TYP_FILE
        .EXTRN  DBG$STA_TYP_RECORD
        .EXTRN  DBG$STA_TYP_SUBRNG
        .EXTRN  DBG$STA_TYP_TYPEDPTR
        .EXTRN  DBG$STA_TYP_VARIANT
        .EXTRN  DBG$STA_TYP_VARIANT_COMP
        .EXTRN  DBG$TYPEID_FOR_ARRAY
        .EXTRN  DBG$TYPEID_FOR_ATOMIC
        .EXTRN  DBG$TYPEID_FOR_SET
        .EXTRN  OTS$CVT_TI_L, DBG$GL_ARRSUB_FLAG
        .EXTRN  DBG$GL_DEVELOPER
        .EXTRN  DBG$GL_CURRENT_PRIMARY
        .EXTRN  DBG$GB_LANGUAGE
        .EXTRN  DBG$GB_MOD_PTR, DBG$GL_ORIG_COMMAND_PTR
        .EXTRN  DBG$GL_RECOMP_FLAG
        .EXTRN  DBG$GB_SET_BREAK_FLAG
        .EXTRN  DBG$GL_UPCASE_COMMAND_PTR
```

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

                        0000 00000 AAA_DUMMY:
                                        .WORD   Save nothing                    ; 3379
                          04 00002      RET                                     ; 3397

; Routine Size:  3 bytes,    Routine Base:  DBG$CODE + 0000
```

```
;  3282          3398  1 GLOBAL ROUTINE DBG$ADDR_EXP_INT(INPUT_DESC, ADDR_EXP_PTR, TYPE, LENGTH,
;  3283          3399  1                                 RADIX, TERM_INDEX) =
;  3284          3400  1 !
;  3285          3401  1 ! FUNCTION
;  3286          3402  1 !       This is the Address Expression Interpreter for most languages
;  3287          3403  1 !       supported by DEBUG.  It parses and evaluates a DEBUG address
;  3288          3404  1 !       expression and returns an Address Expression Descriptor which
;  3289          3405  1 !       represents the value of the expression.  This routine itself is
;  3290          3406  1 !       only a set-up routine which sets up the character pointer and
;  3291          3407  1 !       the expression radix to use and then calls DBG$EXPRESSION_PARSER
;  3292          3408  1 !       to do the actual work.
;  3293          3409  1 !
;  3294          3410  1 ! INPUTS
;  3295          3411  1 !       INPUT_DESC - A string descriptor which points to the input string to be
;  3296          3412  1 !                    parsed as an Address Expression.  Only the pointer field of
;  3297          3413  1 !                    this descriptor is actually used--the string is expected to
;  3298          3414  1 !                    terminated by a carriage-return character.
;  3299          3415  1 !
;  3300          3416  1 !       ADDR_EXP_PTR - The address of a longword location to receive a pointer
;  3301          3417  1 !                      to a Primary or Value Descriptor returned by this routine.
;  3302          3418  1 !
;  3303          3419  1 !       TYPE     - The address of a longword location to receive the address
;  3304          3420  1 !                  "type", namely instruction or not instruction.
;  3305          3421  1 !
;  3306          3422  1 !       LENGTH   - The address of a longword location to receive the length of
;  3307          3423  1 !                  the current instruction if the "type" is instruction.
;  3308          3424  1 !
;  3309          3425  1 !       RADIX    - The radix to be used to interpret integer numbers.  The al-
;  3310          3426  1 !                  lowed radix values are DBG$K_DECIMAL, DBG$K_HEX, DBG$K_OCTAL,
;  3311          3427  1 !                  and DBG$K_BINARY.
;  3312          3428  1 !
;  3313          3429  1 !       TERM_INDEX - A "terminator index" which indicates which lexical tokens
;  3314          3430  1 !                    are allowed as expression terminators in this context.  For
;  3315          3431  1 !                    example, in the EXAMINE command, "," and ":" are allowed
;  3316          3432  1 !                    terminators and in the DEPOSIT command, "=" is the allowed
;  3317          3433  1 !                    terminator.  These index values have names of the form
;  3318          3434  1 !                    TOKEN$K_TERM_xxx.
;  3319          3435  1 !
;  3320          3436  1 ! OUTPUTS
;  3321          3437  1 !       INPUT_DESC - The input string descriptor is updated to point to the
;  3322          3438  1 !                    first character after the address expression just parsed.
;  3323          3439  1 !                    If the parse was stopped by a terminator token, the input
;  3324          3440  1 !                    string descriptor will point to that token on return.
;  3325          3441  1 !
;  3326          3442  1 !       ADDR_EXP_PTR - A Primary or Value Descriptor is constructed and a
;  3327          3443  1 !                      pointer to that descriptor is returned to ADDR_EXP_PTR.
;  3328          3444  1 !
;  3329          3445  1 !       TYPE     - If the address expression yields an instruction address, the
;  3330          3446  1 !                  value DBG$K_INSTRUCTION is returned to TYPE.  Otherwise, the
;  3331          3447  1 !                  value DBG$K_NOTYPE is returned to TYPE.
;  3332          3448  1 !
;  3333          3449  1 !       LENGTH   - If the value DBG$K_INSTRUCTION is returned to TYPE, the
;  3334          3450  1 !                  length in bytes of the instruction pointed to by the address
;  3335          3451  1 !                  expression is returned to LENGTH.  Otherwise, 0 is returned.
;  3336          3452  1 !
;  3337          3453  1 !       The value STS$K_SUCCESS is return as the routine result if the expres-
;  3338          3454  1 !                  sion was terminated by a carriage-return character.  If it
```

```
;  3339          3455  1 !                was terminated any other way (i.e., by a terminator token),
;  3340          3456  1 !                the value STS$K_WARNING is returned.
;  3341          3457  1 !
;  3342          3458  1
;  3343          3459  2     BEGIN
;  3344          3460  2
;  3345          3461  2     MAP
;  3346          3462  2         INPUT_DESC: REF BLOCK[,BYTE],    ! Pointer to input string descriptor
;  3347          3463  2         ADDR_EXP_PTR: REF VECTOR[1],     ! Longword to receive address of a
;  3348          3464  2                                          !      Primary or Value Descriptor
;  3349          3465  2         TYPE: REF VECTOR[1],             ! Longword to receive type code value
;  3350          3466  2         LENGTH: REF VECTOR[1];           ! Longword to receive length value
;  3351          3467  2
;  3352          3468  2
;  3353          3469  2
;  3354          3470  2     ! Set up CHARPTR to point to the start of the expression string.  Also set
;  3355          3471  2     ! up the radix we are to use in the scan and initialize some variables.
;  3356          3472  2     !
;  3357          3473  2     CHARPTR = .INPUT_DESC[DSC$A_POINTER];
;  3358          3474  2     EXPRESSION_RADIX = .RADIX;
;  3359          3475  2     SAVED_TOKEN = 0;
;  3360          3476  2
;  3361          3477  2
;  3362          3478  2     ! Set the address "type" and "length" to indicate "no type".  These values
;  3363          3479  2     ! may be changed to "instruction" if the Expression Parser finds that the
;  3364          3480  2     ! address expression points to an instruction address.  (This allows DEBUG
;  3365          3481  2     ! to automatically display instruction locations as instructions.)
;  3366          3482  2     !
;  3367          3483  2     ADDRESS_TYPE = DBG$K_NOTYPE;
;  3368          3484  2     ADDRESS_LENGTH = 0;
;  3369          3485  2
;  3370          3486  2
;  3371          3487  2     ! Call the Expression Parser to parse the address expression.  Then fix up
;  3372          3488  2     ! the string descriptor to reflect the new location of the parse pointer,
;  3373          3489  2     ! return the address type and length values, and return the appropriate
;  3374          3490  2     ! status code.
;  3375          3491  2     !
;  3376          3492  2     ADDR_EXP_PTR[0] = DBG$EXPRESSION_PARSER(TRUE,
;  3377          3493  2                                             .TERM_POINTER_TBL[.TERM_INDEX] + TABLEBASE);
;  3378          3494  2     INPUT_DESC[DSC$W_LENGTH] = .INPUT_DESC[DSC$W_LENGTH] +
;  3379          3495  2                                             .INPUT_DESC[DSC$A_POINTER] - .CHARPTR;
;  3380          3496  2     INPUT_DESC[DSC$A_POINTER] = .CHARPTR;
;  3381          3497  2     TYPE[0] = .ADDRESS_TYPE;
;  3382          3498  2     LENGTH[0] = .ADDRESS_LENGTH;
;  3383          3499  2     IF .CHARPTR[0] NEQ CAR_RET THEN RETURN STS$K_WARNING;
;  3384          3500  2     RETURN STS$K_SUCCESS;
;  3385          3501  2
;  3386          3502  1     END;


                                  000C 00000          .ENTRY  DBG$ADDR_EXP_INT, Save R2,R3                    ; 3398
                53 00000000'  EF   9E 00002          MOVAB   CHARPTR, R3                                     ;
                52         04 AC   D0 00009          MOVL    INPUT_DESC, R2                                  ; 3473
                63         04 A2   D0 0000D          MOVL    4(R2), CHARPTR                                  ;
```

```
                040C   C3       14   AC  D0 00011          MOVL    RADIX, EXPRESSION_RADIX                 ; 3474
                              0424   C3  D4 00017          CLRL    SAVED_TOKEN                             ; 3475
                  F4   A3       80   8F  9A 0001B          MOVZBL  #128, ADDRESS_TYPE                      ; 3483
                       A3       F0   A3  D4 00020          CLRL    ADDRESS_LENGTH                          ; 3484
                       50       18   AC  D0 00023          MOVL    TERM_INDEX, R0                          ; 3493
                       50 00000000'EF40  D0 00027          MOVL    TERM_POINTER_TBL[R0], R0                ;
                          00000000'EF40  9F 0002F          PUSHAB  TABLEBASE[R0]                           ;
                                     01  DD 00036          PUSHL   #1                                      ; 3492
                0000V  CF              02  FB 00038          CALLS   #2, DBG$EXPRESSION_PARSER               ;
                  08   BC              50  D0 0003D          MOVL    R0, @ADDR_EXP_PTR                       ;
                       51              62  3C 00041          MOVZWL  (R2), R1                                ; 3495
                       51        04    A2  C0 00044          ADDL2   4(R2), R1                               ;
                       50              63  D0 00048          MOVL    CHARPTR, R0                             ;
                62     51              50  A3 0004B          SUBW3   R0, R1, (R2)                            ;
                  04   A2              50  D0 0004F          MOVL    R0, 4(R2)                               ; 3496
                  0C   BC        F4    A3  D0 00053          MOVL    ADDRESS_TYPE, @TYPE                     ; 3497
                  10   BC        F0    A3  D0 00058          MOVL    ADDRESS_LENGTH, @LENGTH                 ; 3498
                       0D              60  91 0005D          CMPB    (R0), #13                               ; 3499
                                       04  12 00060          BNEQ    1$                                      ;
                       50              01  D0 00062          MOVL    #1, R0                                  ; 3500
                                           04 00065          RET                                             ;
                                       50  D4 00066 1$:      CLRL    R0                                      ; 3502
                                           04 00068          RET                                             ;

; Routine Size:  105 bytes,    Routine Base:  DBG$CODE + 0003
```

```
;  3388          3503  1 GLOBAL ROUTINE DBG$BUILD_PRIMARY_SUBNODE(PRIMPTR, KIND, SYMID,
;  3389          3504  1                                          FCODE, TYPEID, DSTPTR): NOVALUE =
;  3390          3505  1 !
;  3391          3506  1 ! FUNCTION
;  3392          3507  1 !       This routine constructs a Primary Descriptor Sub-Node for a specified
;  3393          3508  1 !       symbol and appends this sub-node to a specified Primary Descriptor.
;  3394          3509  1 !       If the symbol is an array or a record, an Array or Record Sub-Node is
;  3395          3510  1 !       constructed and the specific information needed for those data types
;  3396          3511  1 !       is filled in.  Otherwise, a Normal Sub-Node is constructed.
;  3397          3512  1 !
;  3398          3513  1 ! INPUTS
;  3399          3514  1 !       PRIMPTR - A pointer to the Root Node of the Primary Descriptor to
;  3400          3515  1 !                 which the Sub-Node should be appended.
;  3401          3516  1 !
;  3402          3517  1 !       KIND    - The KIND of the symbol for which the Primary Descriptor
;  3403          3518  1 !                 Sub-Node should be constructed.  This is the RST "kind"
;  3404          3519  1 !                 returned by DBG$STA_GETSYMBOL.
;  3405          3520  1 !
;  3406          3521  1 !       SYMID   - The SYMID of the symbol for which the Primary Descriptor
;  3407          3522  1 !                 Sub-Node should be constructed.  If there is no SYMID (as
;  3408          3523  1 !                 for an individual array element, for exmple), SYMID should
;  3409          3524  1 !                 be zero.
;  3410          3525  1 !
;  3411          3526  1 !       FCODE   - The FCODE ("format code") for the data type of the symbol for
;  3412          3527  1 !                 which the Primary Descriptor Sub-Node should be constructed.
;  3413          3528  1 !                 If the symbol is not a data items (i.e., if its KIND is not
;  3414          3529  1 !                 RST$K_DATA or RST$K_TYPCOMP), FCODE should be zero.
;  3415          3530  1 !
;  3416          3531  1 !       TYPEID  - The Type ID of the data item for which the Primary Descriptor
;  3417          3532  1 !                 Sub-Node should be constructed.  If the entity in question is
;  3418          3533  1 !                 not a data item, TYPEID should be zero.
;  3419          3534  1 !
;  3420          3535  1 !       DSTPTR  - A pointer to the DST record corresponding the the data item.
;  3421          3536  1 !                 This is used in the case of BLISS data to obtain the
;  3422          3537  1 !                 information about what kind of BLISS structure this is.
;  3423          3538  1 !
;  3424          3539  1 ! OUTPUTS
;  3425          3540  1 !       A Primary Descriptor Sub-Node is created and appended to the PRIMPTR
;  3426          3541  1 !                 Primary Descriptor.  PRIMPTR itself is not changed, however.
;  3427          3542  1 !                 There is no other output.
;  3428          3543  1 !
;  3429          3544  1
;  3430          3545  2     BEGIN
;  3431          3546  2
;  3432          3547  2     MAP
;  3433          3548  2         DSTPTR: REF DST$RECORD,          ! Pointer to DST record
;  3434          3549  2         PRIMPTR: REF DBG$PRIMARY,        ! Pointer to Primary Descriptor
;  3435          3550  2         SYMID: REF RST$ENTRY;            ! Pointer to symbol's RST entry
;  3436          3551  2
;  3437          3552  2     BUILTIN
;  3438          3553  2         INSQUE;                          ! Insert-Queue function
;  3439          3554  2
;  3440          3555  2     LITERAL
;  3441          3556  2         MAX_DIMS = 20;                   ! Maximum dimension count we allow
;  3442          3557  2
;  3443          3558  2     LOCAL
;  3444          3559  2         ATOMIC_TYPE,                     ! A dtype code
```

```
;3445     3560  2          BIT_LENGTH,                         ! Array element bit length
;3446     3561  2          BITSIZE,                            ! Array or record component bit-size
;3447     3562  2                                              !     (not actually used)
;3448     3563  2          BOUNDVEC: REF VECTOR[,LONG],        ! Pointer to bounds vector in array
;3449     3564  2                                              !     descriptor
;3450     3565  2          CELLTYPE,                           ! TYPEID for array element cell type
;3451     3566  2          COMP_VEC,                           ! Record Component vector (not used)
;3452     3567  2          DIMVECPTR: REF VECTOR[,LONG],       ! Pointer to vector of subscript TYPEIDs
;3453     3568  2          DSCADDR: REF BLOCK[,BYTE],          ! Pointer to array descriptor
;3454     3569  2          HIGHPTR,                            ! Higher bound of the subrange value
;3455     3570  2          LENGTH,                             ! Byte length of array element
;3456     3571  2          LOWPTR,                             ! Lower bound of the subrange value
;3457     3572  2          NCOMPS,                             ! Number of record components (not used)
;3458     3573  2          NDIMS,                              ! Number of array dimensions
;3459     3574  2          NODEPTR: REF DBG$PRIM_NODE,         ! Pointer to created Primary Sub-Node
;3460     3575  2          OFFSET,                             ! Computed offset for ARRAY[0,0,...,0]
;3461     3576  2          SIZE,                               ! Size in bits of the data type
;3462     3577  2          STRIDE: VECTOR[MAX_DIMS, LONG],     ! Vector of strides computed from
;3463     3578  2                                              !     multipliers in array descriptor
;3464     3579  2          STRIDEPTR: REF VECTOR[,LONG],       ! Pointer to stride or multiplier
;3465     3580  2                                              !     vector in array descriptor
;3466     3581  2          STRIDE_SIZE,                        ! Size of stride for BLISS vectors
;3467     3582  2          SUB_TYPEID: REF RST$ENTRY,          ! Pointer to TYPEID RST entry
;3468     3583  2          SUBVECTOR:                          ! Pointer to subscript block-vector
;3469     3584  2                  REF DBG$PRIM_NODE_SUBS,     !     in Primary Descr Array Sub-Node
;3470     3585  2          TMP_SYMID: REF RST$ENTRY;           ! Scratch SYMID
;3471     3586  2
;3472     3587  2      LABEL
;3473     3588  2          COMPUTE_STRIDES;
;3474     3589  2
;3475     3590  2
;3476     3591  2      DBG$GL_CURRENT_PRIMARY = .PRIMPTR;
;3477     3592  2
;3478     3593  2      ! Determine what kind of symbol this is and allocate the Sub-Node memory
;3479     3594  2      ! block accordingly.  If this symbol is an array or a record, we also must
;3480     3595  2      ! collect the special information needed for those data types and fill it
;3481     3596  2      ! into the Primary Descriptor Sub-Node.
;3482     3597  2      !
;3483     3598  2      CASE .FCODE FROM RST$K_TYPE_MINIMUM TO RST$K_TYPE_MAXIMUM OF
;3484     3599  2          SET
;3485     3600  2
;3486     3601  2
;3487     3602  2          ! Handle arrays.  Determine the number of dimensions in the array and
;3488     3603  2          ! allocate an Array Sub-Node large enough to hold that many dimensions.
;3489     3604  2          ! Then fill in all the fields specific to the Array Sub-Node.
;3490     3605  2          !
;3491     3606  2          [RST$K_TYPE_ARRAY]:
;3492     3607  3              BEGIN
;3493     3608  3
;3494     3609  3
;3495     3610  3              ! Get all needed type information about the array type from
;3496     3611  3              ! DBG$STA_TYP_ARRAY.  Allocate a memory block for the Array
;3497     3612  3              ! Sub-Node and fill in the fields in the fixed part of that
;3498     3613  3              ! Sub-Node.
;3499     3614  3              !
;3500     3615  3              DBG$STA_TYP_ARRAY(.TYPEID, DSCADDR, CELLTYPE,
;3501     3616  3                                                  NDIMS, DIMVECPTR, BITSIZE);
```

```
;  3502          3617  3
;  3503          3618  3
;  3504          3619  3          ! If we got a symid passed in to this routine, and the symid
;  3505          3620  3          ! represents an array, try calling SYMVALUE to get an array
;  3506          3621  3          ! descriptor for this array. If we get one, then use this
;  3507          3622  3          ! descriptor instead of the one we got back from STA_TYP_ARRAY.
;  3508          3623  3          !
;  3509          3624  3          ! Note - normally, these 2 descriptors will be the same.
;  3510          3625  3          ! However, for dynamic arrays in PASCAL, the runtime descriptor
;  3511          3626  3          ! (which we get back when we call SYMVALUE with the symid) is
;  3512          3627  3          ! correct, but the compile-time descriptor (which is part of
;  3513          3628  3          ! the typespec) is wrong. This code is a workaround for this
;  3514          3629  3          ! problem in the PASCAL DST.
;  3515          3630  3          !
;  3516          3631  3          IF .SYMID NEQ 0
;  3517          3632  3          THEN
;  3518          3633  4              BEGIN
;  3519          3634  4              LOCAL
;  3520          3635  4                  DESC: VECTOR[3],
;  3521          3636  4                  RSTPTR: REF RST$ENTRY,
;  3522          3637  4                  VALUE_KIND;
;  3523          3638  4              RSTPTR = .SYMID;
;  3524          3639  4              WHILE .RSTPTR[RST$B_KIND] NEQ RST$K_MODULE DO
;  3525          3640  4                  RSTPTR = .RSTPTR[RST$L_UPSCOPEPTR];
;  3526          3641  4              IF .RSTPTR[RST$B_LANGUAGE] EQL DBG$K_PASCAL
;  3527          3642  4              THEN
;  3528          3643  5                  BEGIN
;  3529          3644  5                  DBG$STA_SETCONTEXT(.SYMID);
;  3530          3645  5                  DBG$STA_SYMVALUE(.SYMID, DESC, VALUE_KIND);
;  3531          3646  5                  IF .VALUE_KIND EQL DBG$K_VAL_DESCR
;  3532          3647  5                  THEN
;  3533          3648  5                      DSCADDR = .DESC[0];
;  3534          3649  4                  END;
;  3535          3650  3              END;
;  3536          3651  3
;  3537          3652  3          IF .NDIMS GTR MAX_DIMS THEN SIGNAL(DBG$_TOOMANDIM);
;  3538          3653  3          NODEPTR = DBG$GET_TEMPMEM(DBG$K_PRIM_SIZE_ARRAY +
;  3539          3654  3                                    .NDIMS*DBG$K_PRIM_SIZE_SUBS);
;  3540          3655  3          NODEPTR[DBG$B_PNARR_DIMCNT] = .NDIMS;
;  3541          3656  3
;  3542          3657  3
;  3543          3658  3          !*** The following is a temporary workaround to a problem in
;  3544          3659  3          !*** the PASCAL DST: They are giving us array descriptors with
;  3545          3660  3          !*** class UBA but dtype=0. Since dtype must be VU for this
;  3546          3661  3          !*** class, we fill in the correct dtype here in this case.
;  3547          3662  3          !
;  3548          3663  3          IF .DSCADDR[DSC$B_CLASS] EQL DSC$K_CLASS_UBA
;  3549          3664  3          THEN
;  3550          3665  3              NODEPTR[DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_VU
;  3551          3666  3
;  3552          3667  3          ELSE
;  3553          3668  3          !*** End temporary workaround.
;  3554          3669  3
;  3555          3670  3              NODEPTR[DBG$B_PNARR_DTYPE] = .DSCADDR[DSC$B_DTYPE];
;  3556          3671  3
;  3557          3672  3          NODEPTR[DBG$W_PNARR_LENGTH] = .DSCADDR[DSC$W_LENGTH];
;  3558          3673  3          NODEPTR[DBG$B_PNARR_SCALE] = .DSCADDR[DSC$B_SCALE];
```

```
;  3559          3674  3          NODEPTR[DBG$B_PNARR_DIGITS] = .DSCADDR[DSC$B_DIGITS];
;  3560          3675  3          NODEPTR[DBG$V_PNARR_COLUMN] = .DSCADDR[DSC$V_FL_COLUMN];
;  3561          3676  3          NODEPTR[DBG$L_PNARR_CELLTYPE] = .CELLTYPE;
;  3562          3677  3
;  3563          3678  3
;  3564          3679  3          ! Set up pointers to the stride (or multiplier) vector and to the
;  3565          3680  3          ! bounds vector in the array descriptor.
;  3566          3681  3          !
;  3567          3682  3          STRIDEPTR = .DSCADDR + 20;
;  3568          3683  3          BOUNDVEC = .STRIDEPTR + 4*.NDIMS;
;  3569          3684  3
;  3570          3685  3
;  3571          3686  3          ! Determine what kind of array descriptor we have.  Determine
;  3572          3687  3          ! whether we have multipliers or strides and whether we are
;  3573          3688  3          ! doing byte or bit addressing.
;  3574          3689  3          !
;  3575          3690  3          CASE .DSCADDR[DSC$B_CLASS] FROM DSC$K_CLASS_S TO DSC$K_CLASS_UBA OF
;  3576          3691  3              SET
;  3577          3692  3
;  3578          3693  3
;  3579          3694  3              ! Handle an ordinary Array Descriptor.  This descriptor has
;  3580          3695  3              ! byte multipliers.  Since we only allow strides in the Array
;  3581          3696  3              ! Sub-Node, we convert the array multipliers into strides here.
;  3582          3697  3              ! (Note that STRIDEPTR points to multipliers in this case.)
;  3583          3698  3              !
;  3584          3699  3              [DSC$K_CLASS_A]:
;  3585          3700  4                  COMPUTE_STRIDES: BEGIN
;  3586          3701  4
;  3587          3702  4
;  3588          3703  4                  !*** Temporary workaround to a problem in the PL/I DST.
;  3589          3704  4                  !*** The multipliers that we are getting in the array
;  3590          3705  4                  !*** descriptors are not correct. They are giving us
;  3591          3706  4                  !*** strides instead of multipliers. So we skip the
;  3592          3707  4                  !*** computation of strides if language is PL/I.
;  3593          3708  4                  !
;  3594          3709  4                  TMP_SYMID = .PRIMPTR[DBG$L_DHDR_SYMIDO];
;  3595          3710  4                  IF .TMP_SYMID NEQ 0
;  3596          3711  4                  THEN
;  3597          3712  5                      BEGIN
;  3598          3713  5                      WHILE .TMP_SYMID[RST$B_KIND] NEQ RST$K_MODULE DO
;  3599          3714  5                          TMP_SYMID = .TMP_SYMID[RST$L_UPSCOPEPTR];
;  3600          3715  5                      IF (.TMP_SYMID[RST$B_LANGUAGE] EQL DBG$K_PLI) AND
;  3601          3716  6                         (.TMP_SYMID[RST$V_OLDPLIFLAG])
;  3602          3717  5                      THEN
;  3603          3718  5                          LEAVE COMPUTE_STRIDES;
;  3604          3719  4                      END;
;  3605          3720  4
;  3606          3721  4
;  3607          3722  4                  ! Pick up the array element length in bytes.  We need this
;  3608          3723  4                  ! length to compute strides properly below.
;  3609          3724  4                  !
;  3610          3725  4                  LENGTH = DBG$DATA_LENGTH(.DSCADDR);
;  3611          3726  4                  LENGTH = (.LENGTH + 7)/8;
;  3612          3727  4
;  3613          3728  4
;  3614          3729  4                  ! If this is a column-major order array, we compute the
;  3615          3730  4                  ! array's stride values from its multiplier values.
```

```
;  3616     3731  4                !
;  3617     3732  4                IF .DSCADDR[DSC$V_FL_COLUMN]
;  3618     3733  4                THEN
;  3619     3734  5                    BEGIN
;  3620     3735  5                    STRIDE[0] = .LENGTH;
;  3621     3736  5                    INCR I FROM 1 TO .NDIMS - 1 DO
;  3622     3737  5                        STRIDE[.I] = .STRIDE[.I - 1]*.STRIDEPTR[.I - 1];
;  3623     3738  5
;  3624     3739  5                    END
;  3625     3740  5
;  3626     3741  5
;  3627     3742  5                ! If this is a row-major order array, we compute the
;  3628     3743  5                ! strides in the opposite direction.
;  3629     3744  5                !
;  3630     3745  4                ELSE
;  3631     3746  5                    BEGIN
;  3632     3747  5                    STRIDE[.NDIMS - 1] = .LENGTH;
;  3633     3748  5                    DECR I FROM .NDIMS - 2 TO 0 DO
;  3634     3749  5                        STRIDE[.I] = .STRIDE[.I + 1]*.STRIDEPTR[.I + 1];
;  3635     3750  5
;  3636     3751  4                    END;
;  3637     3752  4
;  3638     3753  4
;  3639     3754  4                ! Make STRIDEPTR point to the newly computed stride vector.
;  3640     3755  4                !
;  3641     3756  4                STRIDEPTR = STRIDE[0];
;  3642     3757  3                END;
;  3643     3758  3
;  3644     3759  3
;  3645     3760  3                ! Handle a Noncontiguous Array Descriptor or a Varying String
;  3646     3761  3                ! Descriptor.  Here we already have byte strides instead of
;  3647     3762  3                ! multipliers so there is nothing we need to do.
;  3648     3763  3                !
;  3649     3764  3                [DSC$K_CLASS_NCA,
;  3650     3765  3                 DSC$K_CLASS_VSA]:
;  3651     3766  3                    0;
;  3652     3767  3
;  3653     3768  3
;  3654     3769  3                ! Handle an Unaligned Bit Array Descriptor.  This descriptor
;  3655     3770  3                ! is like the Noncontiguous Array Descriptor except that it has
;  3656     3771  3                ! bit strides instead of byte strides.
;  3657     3772  3                !
;  3658     3773  3                [DSC$K_CLASS_UBA]:
;  3659     3774  3                    NODEPTR[DBG$V_PNARR_BITREF] = TRUE;
;  3660     3775  3
;  3661     3776  3
;  3662     3777  3                ! Any other case constitutes an invalid array descriptor.
;  3663     3778  3                ! Signal an error message.
;  3664     3779  3                !
;  3665     3780  3                [INRANGE, OUTRANGE]:
;  36  ,    3781  3                    SIGNAL(DBG$_INVARRDSC);
;  3667     3782  3
;  3668     3783  3                TES;
;  3669     3784  3
;  3670     3785  3
;  3671     3786  3            ! Loop through the dimensions of the array to set up the subscript
;  3672     3787  3            ! block-vector in the Array Sub-Node.  In that vector, we set the
```

```
;  3673          3788  3          ! subscript value to be the same as the lower bound--this is changed
;  3674          3789  3          ! later when the actual subscript value is picked up.  The lower and
;  3675          3790  3          ! upper bounds, the stride, and the subscript type TYPEID are all
;  3676          3791  3          ! picked up and stored away as well.
;  3677          3792  3          !
;  3678          3793  3          SUBVECTOR = NODEPTR[DBG$A_PNARR_SVECTOR];
;  3679          3794  3          OFFSET = 0;
;  3680          3795  3          INCR I FROM 0 TO .NDIMS - 1 DO
;  3681          3796  4              BEGIN
;  3682          3797  4              SUBVECTOR[.I, DBG$L_PNSUB_SVALUE] = .BOUNDVEC[2*.I];
;  3683          3798  4              SUBVECTOR[.I, DBG$L_PNSUB_STRIDE] = .STRIDEPTR[.I];
;  3684          3799  4              SUBVECTOR[.I, DBG$L_PNSUB_LBOUND] = .BOUNDVEC[2*.I];
;  3685          3800  4              SUBVECTOR[.I, DBG$L_PNSUB_UBOUND] = .BOUNDVEC[2*.I + 1];
;  3686          3801  4
;  3687          3802  4              SUB_TYPEID = 0;
;  3688          3803  4              IF .DIMVECPTR[.I] NEQ 0
;  3689          3804  4              THEN
;  3690          3805  5                  BEGIN
;  3691          3806  5                  SUB_TYPEID = .DIMVECPTR[.I];
;  3692          3807  5                  WHILE .SUB_TYPEID[RST$B_FCODE] EQL RST$K_TYPE_SUBRNG DO
;  3693          3808  5                      DBG$STA_TYP_SUBRNG(.SUB_TYPEID,SUB_TYPEID,LOWPTR,HIGHPTR,SIZE);
;  3694          3809  5
;  3695          3810  5                  SUBVECTOR[.I, DBG$L_PNSUB_TYPEID] = .SUB_TYPEID;
;  3696          3811  5                  END
;  3697          3812  5
;  3698          3813  4              ELSE
;  3699          3814  4                  SUBVECTOR[.I, DBG$L_PNSUB_TYPEID] = .DIMVECPTR[.I];
;  3700          3815  4
;  3701          3816  5              OFFSET = .OFFSET - (.SUBVECTOR[.I, DBG$L_PNSUB_STRIDE]*
;  3702          3817  4                                  .BOUNDVEC[2*.I]);
;  3703          3818  4
;  3704          3819  4
;  3705          3820  4              ! One additional thing needs to be done for arrays indexed
;  3706          3821  4              ! by enumeration types in ADA. The SVALUE field needs
;  3707          3822  4              ! to be filled in to be the value of the enumeration,
;  3708          3823  4              ! not its position.
;  3709          3824  4              !
;  3710          3825  4              IF .DBG$GB_LANGUAGE EQL DBG$K_ADA
;  3711          3826  4              THEN
;  3712          3827  4                  IF .SUB_TYPEID NEQ 0
;  3713          3828  4                  THEN
;  3714          3829  4                      IF .SUB_TYPEID[RST$B_FCODE] EQL RST$K_TYPE_ENUM
;  3715          3830  4                      THEN
;  3716          3831  4                          SUBVECTOR[.I, DBG$L_PNSUB_SVALUE] =
;  3717          3832  4                              DBG$ENUM_VAL(.SUB_TYPEID,
;  3718          3833  4                                  .SUBVECTOR[.I, DBG$L_PNSUB_SVALUE]);
;  3719          3834  4
;  3720          3835  3              END;
;  3721          3836  3
;  3722          3837  3
;  3723          3838  3          ! Finally fill in the offset from the start of the array to
;  3724          3839  3          ! element ARRAY[0,0,...,0].  Also mark symbol as an aggregate.
;  3725          3840  3          !
;  3726          3841  3          NODEPTR[DBG$L_PNARR_OFFSET] = .OFFSET;
;  3727          3842  3          IF .DBG$GL_ARRSUB_FLAG
;  3728          3843  3          THEN
;  3729          3844  3              PRIMPTR[DBG$V_DHDR_AGGR] = TRUE;
```

```
;  3730     3845  3
;  3731     3846  2              END;                            ! End of case for Array Sub-Node
;  3732     3847  2
;  3733     3848  2
;  3734     3849  2          ! Handle records.  Allocate a Record Sub-Node.  Then simply initialize
;  3735     3850  2          ! the record component index to be 1.  If dot-qualification follows,
;  3736     3851  2          ! this field will be changed to the actual component index of the
;  3737     3852  2          ! selected record component.  Also mark symbol as an aggregate.
;  3738     3853  2          !
;  3739     3854  2          [RST$K_TYPE_RECORD]:
;  3740     3855  3              BEGIN
;  3741     3856  3              DBG$STA_TYP_RECORD(.TYPEID, NCOMPS, COMP_VEC, BITSIZE);
;  3742     3857  3              NODEPTR = DBG$GET_TEMPMEM(DBG$K_PRIM_SIZE_RECORD);
;  3743     3858  3              NODEPTR[DBG$W_PNREC_INDEX] = 1;
;  3744     3859  3              NODEPTR[DBG$W_PNREC_NCOMPS] = .NCOMPS;
;  3745     3860  3              IF .DBG$GL_RECCMP_FLAG
;  3746     3861  3              THEN
;  3747     3862  3                  PRIMPTR[DBG$V_DHDR_AGGR] = TRUE;
;  3748     3863  2              END;
;  3749     3864  2
;  3750     3865  2
;  3751     3866  2          ! Handle Record Variants (as in PASCAL or ADA).  Here we allocate
;  3752     3867  2          ! a Variant Sub-Node, but the fields specific to this sub-node are
;  3753     3868  2          ! actually set by the caller (namely GET_RECORD_COMPONENT).
;  3754     3869  2          !
;  3755     3870  2          [RST$K_TYPE_VARIANT]:
;  3756     3871  2              NODEPTR = DBG$GET_TEMPMEM(DBG$K_PRIM_SIZE_VARIANT);
;  3757     3872  2
;  3758     3873  2
;  3759     3874  2          ! Handle BLISS data items. The four kinds of data that fall have this
;  3760     3875  2          ! fcode are vectors, bitvectors, blocks, and blockvectors. For these
;  3761     3876  2          ! four types, we have to go to the DST to obtain the information
;  3762     3877  2          ! which is placed in the Primary Descriptor.
;  3763     3878  2          !
;  3764     3879  2          [RST$K_TYPE_BLIDATA]:
;  3765     3880  3              BEGIN
;  3766     3881  3
;  3767     3882  3
;  3768     3883  3              ! Assume aggregate until proved otherwise.
;  3769     3884  3              !
;  3770     3885  3              PRIMPTR [DBG$V_DHDR_AGGR] = TRUE;
;  3771     3886  3
;  3772     3887  3
;  3773     3888  3              ! Get the DST record if necessary.
;  3774     3889  3              !
;  3775     3890  3              IF .DSTPTR EQL 0 AND .SYMID NEQ 0
;  3776     3891  3              THEN
;  3777     3892  3                  DSTPTR = .SYMID [RST$L_DSTPTR];
;  3778     3893  3
;  3779     3894  3
;  3780     3895  3              ! Check for the REF bit being set.  If so, allocate a Primary
;  3781     3896  3              ! Descriptor Normal Sub-Node and set the FCODE to pointer to
;  3782     3897  3              ! indicate that dereferencing is taking place.  Also turn off
;  3783     3898  3              ! the aggregate flag for REF items.
;  3784     3899  3              !
;  3785     3900  3              IF .DSTPTR [DST$V_BLI_REF] AND .SYMID NEQ 0
;  3786     3901  3              THEN
```

```
;  3787          3902  4                BEGIN
;  3788          3903  4                NODEPTR = DBG$GET_TEMPMEM (DBG$K_PRIM_SIZE_NORMAL);
;  3789          3904  4                PRIMPTR [DBG$V_DHDR_AGGR] = FALSE;
;  3790          3905  4                FCODE = RST$K_TYPE_PTR;
;  3791          3906  4                END
;  3792          3907  4
;  3793          3908  4
;  3794          3909  4            ! If it is not a REF item, CASE on the kind of BLISS structure:
;  3795          3910  4            ! Vector, Bitvector, Block, or Blockvector.
;  3796          3911  4            !
;  3797          3912  3            ELSE
;  3798          3913  4                BEGIN
;  3799          3914  4                CASE .DSTPTR [DST$V_BLI_STRUC] FROM DST$K_BLI_NOSTRUC
;  3800          3915  4                                               TO   DST$K_BLI_BLKVEC OF
;  3801          3916  4                    SET
;  3802          3917  4
;  3803          3918  4
;  3804          3919  4                    ! The "nostruc" code arises in the case where the structure of
;  3805          3920  4                    ! the data is a user-defined structure, and not one of the
;  3806          3921  4                    ! four built-ins. We treat such data as an ordinary variable.
;  3807          3922  4                    !
;  3808          3923  4                    [DST$K_BLI_NOSTRUC] :
;  3809          3924  5                        BEGIN
;  3810          3925  5                        PRIMPTR [DBG$V_DHDR_AGGR] = FALSE;
;  3811          3926  5                        NODEPTR = DBG$GET_TEMPMEM (DBG$K_PRIM_SIZE_NORMAL);
;  3812          3927  4                        END;
;  3813          3928  4
;  3814          3929  4
;  3815          3930  4                    ! Handle vectors.  We allocate enough space for an Array
;  3816          3931  4                    ! Sub-Node with one subscript.  The fields of the subnode
;  3817          3932  4                    ! are then filled in.
;  3818          3933  4                    !
;  3819          3934  4                    [DST$K_BLI_VEC] :
;  3820          3935  5                        BEGIN
;  3821          3936  5                        FCODE = RST$K_TYPE_ARRAY;
;  3822          3937  5                        NODEPTR = DBG$GET_TEMPMEM (DBG$K_PRIM_SIZE_ARRAY +
;  3823          3938  5                                                   DBG$K_PRIM_SIZE_SUBS);
;  3824          3939  5                        NODEPTR [DBG$B_PNARR_DIMCNT] = 1;
;  3825          3940  5                        NODEPTR [DBG$W_PNARR_LENGTH] =  .DSTPTR[DST$V_BLI_VEC_UNIT_SIZE];
;  3826          3941  5
;  3827          3942  5                        !+
;  3828          3943  5                        ! Figure out the dtype based on the unit size and
;  3829          3944  5                        ! sign extension.
;  3830          3945  5                        !-
;  3831          3946  5
;  3832          3947  5                        ! VECTOR[BYTE] or VECTOR[BYTE,SIGNED]
;  3833          3948  5                        !
;  3834          3949  5                        IF .DSTPTR[DST$V_BLI_VEC_UNIT_SIZE] EQL 1
;  3835          3950  5                        THEN
;  3836          3951  5                            IF .DSTPTR[DST$V_BLI_VEC_SIGN_EXT] NEQ 0
;  3837          3952  5                            THEN
;  3838          3953  5                                NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_B
;  3839          3954  5                            ELSE
;  3840          3955  5                                NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_BU
;  3841          3956  5
;  3842          3957  5                        ! VECTOR[WORD] or VECTOR[WORD,SIGNED]
;  3843          3958  5                        !
```

```
;  3844          3959  5                ELSE IF .DSTPTR[DST$V_BLI_VEC_UNIT_SIZE] EQL 2
;  3845          3960  5                THEN
;  3846          3961  5                    IF .DSTPTR[DST$V_BLI_VEC_SIGN_EXT] NEQ 0
;  3847          3962  5                    THEN
;  3848          3963  5                        NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_W
;  3849          3964  5                    ELSE
;  3850          3965  5                        NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_WU
;  3851          3966  5
;  3852          3967  5                ! VECTOR[LONG] or VECTOR[LONG,SIGNED]
;  3853          3968  5                !
;  3854          3969  5                ELSE IF .DSTPTR[DST$V_BLI_VEC_UNIT_SIZE] EQL 4
;  3855          3970  5                THEN
;  3856          3971  5                    IF .DSTPTR[DST$V_BLI_VEC_SIGN_EXT] NEQ 0
;  3857          3972  5                    THEN
;  3858          3973  5                        NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_L
;  3859          3974  5                    ELSE
;  3860          3975  5                        NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_LU
;  3861          3976  5
;  3862          3977  5                ! VECTOR[X] where X is not 1, 2, or 4
;  3863          3978  5                !
;  3864          3979  5                ELSE
;  3865          3980  6                    BEGIN
;  3866          3981  6                    IF .DSTPTR[DST$V_BLI_VEC_SIGN_EXT] NEQ 0
;  3867          3982  6                    THEN
;  3868          3983  6                        NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_V
;  3869          3984  6                    ELSE
;  3870          3985  6                        NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_VU;
;  3871          3986  6
;  3872          3987  6                    ! In this case the since the type has been set
;  3873          3988  6                    ! to bitfield then multiply the length by 8 to
;  3874          3989  6                    ! express it in bits.
;  3875          3990  6                    !
;  3876          3991  6                    NODEPTR[DBG$W_PNARR_LENGTH] =
;  3877          3992  6                        .NODEPTR[DBG$W_PNARR_LENGTH] * 8;
;  3878          3993  5                    END;
;  3879          3994  5
;  3880          3995  5                SUBVECTOR = NODEPTR [DBG$A_PNARR_SVECTOR];
;  3881          3996  5
;  3882          3997  5
;  3883          3998  5                ! The stride can be either 1, 2, or 4, depending on whether
;  3884          3999  5                ! the vector was declared as a byte, word, or longword
;  3885          4000  5                ! vector. We obtain this information from the DST.
;  3886          4001  5                !
;  3887          4002  5                SUBVECTOR [0, DBG$L_PNSUB_STRIDE] =
;  3888          4003  5                    .DSTPTR [DST$V_BLI_VEC_UNIT_SIZE];
;  3889          4004  5
;  3890          4005  5
;  3891          4006  5                ! The upper bound on subscripts is one less than the
;  3892          4007  5                ! number of units that were allocated in the declaration
;  3893          4008  5                ! of the vector. (Origin-0 subscripting)
;  3894          4009  5                !
;  3895          4010  5                IF .DSTPTR [DST$L_BLI_VEC_UNITS] EQL 0
;  3896          4011  5                THEN
;  3897          4012  5                    SUBVECTOR [0, DBG$L_PNSUB_UBOUND] = 0
;  3898          4013  5
;  3899          4014  5                ELSE
;  3900          4015  5                    SUBVECTOR [0, DBG$L_PNSUB_UBOUND] =
```

```
;  3901          4016  5                    .DSTPTR [DST$L_BLI_VEC_UNITS]-1;
;  3902          4017  5
;  3903          4018  4                END;
;  3904          4019  4
;  3905          4020  4
;  3906          4021  4            ! Handle bit-vectors.  We allocate enough space for an
;  3907          4022  4            ! Array Sub-Node with one subscript.  The fields of the
;  3908          4023  4            ! subnode are then filled in.
;  3909          4024  4            !
;  3910          4025  4            [DST$K_BLI_BITVEC] :
;  3911          4026  5                BEGIN
;  3912          4027  5                FCODE = RST$K_TYPE_ARRAY;
;  3913          4028  5                NODEPTR = DBG$GET_TEMPMEM (DBG$K_PRIM_SIZE_ARRAY +
;  3914          4029  5                                           DBG$K_PRIM_SIZE_SUBS);
;  3915          4030  5
;  3916          4031  5
;  3917          4032  5                ! Since this is a bitvector, set the BITREF flag.
;  3918          4033  5                !
;  3919          4034  5                NODEPTR [DBG$V_PNARR_BITREF] = TRUE;
;  3920          4035  5                NODEPTR [DBG$B_PNARR_DIMCNT] = 1;
;  3921          4036  5                NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_VU;
;  3922          4037  5                NODEPTR [DBG$W_PNARR_LENGTH] = 1;
;  3923          4038  5                SUBVECTOR = NODEPTR [DBG$A_PNARR_SVECTOR];
;  3924          4039  5                SUBVECTOR [0, DBG$L_PNSUB_STRIDE] = 1;
;  3925          4040  5
;  3926          4041  5
;  3927          4042  5                ! The upper bound on subscripts is one less than the
;  3928          4043  5                ! number of units that were allocated in the declaration
;  3929          4044  5                ! of the vector. (Origin-0 subscripting)
;  3930          4045  5                !
;  3931          4046  5                IF .DSTPTR [DST$L_BLI_BITVEC_SIZE] EQL 0
;  3932          4047  5                THEN
;  3933          4048  5                    SUBVECTOR [0, DBG$L_PNSUB_UBOUND] = 0
;  3934          4049  5
;  3935          4050  5                ELSE
;  3936          4051  5                    SUBVECTOR [0, DBG$L_PNSUB_UBOUND] =
;  3937          4052  5                        .DSTPTR [DST$L_BLI_BITVEC_SIZE]-1;
;  3938          4053  5
;  3939          4054  4                END;
;  3940          4055  4
;  3941          4056  4
;  3942          4057  4            ! Handle blocks.  We allocate enough space for an Array
;  3943          4058  4            ! Sub-Node with one subscript.  The fields of the subnode
;  3944          4059  4            ! are then filled in.
;  3945          4060  4            !
;  3946          4061  4            [DST$K_BLI_BLOCK] :
;  3947          4062  5                BEGIN
;  3948          4063  5                PRIMPTR[DBG$V_DMDR_BLIBLK] = TRUE;
;  3949          4064  5                FCODE = RST$K_TYPE_ARRAY;
;  3950          4065  5                NODEPTR = DBG$GET_TEMPMEM (DBG$K_PRIM_SIZE_ARRAY +
;  3951          4066  5                                           DBG$K_PRIM_SIZE_SUBS);
;  3952          4067  5                NODEPTR [DBG$B_PNARR_DIMCNT] = 1;
;  3953          4068  5                SUBVECTOR = NODEPTR [DBG$A_PNARR_SVECTOR];
;  3954          4069  5
;  3955          4070  5
;  3956          4071  5                ! Fill in stride and length as four. For purposes
;  3957          4072  5                ! of aggregate examine, we are representing
```

```
;  3958          4073  5                    ! each element as a longword. If it turns out
;  3959          4074  5                    ! we are not doing aggregate examine, then
;  3960          4075  5                    ! we fix up this information in the GET_BLISS_SUBSCRIPTS
;  3961          4076  5                    ! routine.
;  3962          4077  5                    !
;  3963          4078  5                    SUBVECTOR [0, DBG$L_PNSUB_STRIDE] = 4;
;  3964          4079  5                    NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_L;
;  3965          4080  5                    NODEPTR [DBG$W_PNARR_LENGTH] = 4;
;  3966          4081  5
;  3967          4082  5
;  3968          4083  5                    ! The upper bound on subscripts is one less than the
;  3969          4084  5                    ! number of units that were allocated in the declaration
;  3970          4085  5                    ! of the vector. (Origin-0 subscripting)
;  3971          4086  5                    ! We temporarily dummy this up as if stride were 4, for
;  3972          4087  5                    ! purposes of aggregate output.
;  3973          4088  5                    !
;  3974          4089  5                    STRIDE_SIZE = .DSTPTR [DST$V_BLI_BLOCK_UNIT_SIZE];
;  3975          4090  5                    IF (.DSTPTR [DST$L_BLI_BLOCK_UNITS] EQL 0) OR
;  3976          4091  6                      (.STRIDE_SIZE EQL 0)
;  3977          4092  5                    THEN
;  3978          4093  5                        SUBVECTOR [0, DBG$L_PNSUB_UBOUND] = 0
;  3979          4094  5
;  3980          4095  5                    ELSE
;  3981          4096  5                        SUBVECTOR [0, DBG$L_PNSUB_UBOUND] =
;  3982          4097  6                          (.DSTPTR [DST$L_BLI_BLOCK_UNITS]*.STRIDE_SIZE-1)
;  3983          4098  5                                                  /4;
;  3984          4099  5
;  3985          4100  4                    END;
;  3986          4101  4
;  3987          4102  4
;  3988          4103  4                ! Handle blockvectors.  We allocate enough space for an
;  3989          4104  4                ! Array Sub-Node with one subscript. The fields of the
;  3990          4105  4                ! subnode are then filled in.
;  3991          4106  4                !
;  3992          4107  4                [DST$K_BLI_BLKVEC] :
;  3993          4108  5                    BEGIN
;  3994          4109  5                    PRIMPTR[DBG$V_DHDR_BLIBLK] = TRUE;
;  3995          4110  5                    FCODE = RST$K_TYPE_ARRAY;
;  3996          4111  5                    NODEPTR = DBG$GET_TEMPMEM (DBG$K_PRIM_SIZE_ARRAY +
;  3997          4112  5                                2 * DBG$K_PRIM_SIZE_SUBS);
;  3998          4113  5                    NODEPTR [DBG$B_PNARR_DIMCNT] = 2;
;  3999          4114  5                    NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_V;
;  4000          4115  5                    NODEPTR [DBG$W_PNARR_LENGTH] = 32;
;  4001          4116  5                    SUBVECTOR = NODEPTR [DBG$A_PNARR_SVECTOR];
;  4002          4117  5
;  4003          4118  5
;  4004          4119  5                    ! The stride on the first subscript is the stride on
;  4005          4120  5                    ! the second subscript times the number of units per
;  4006          4121  5                    ! block. The upper bound on the first subscript
;  4007          4122  5                    ! depends on the number of blocks in the blockvector.
;  4008          4123  5                    !
;  4009          4124  5                    SUBVECTOR [0, DBG$L_PNSUB_STRIDE] =
;  4010          4125  5                                    .DSTPTR [DST$B_BLI_BLKVEC_UNIT_SIZE] *
;  4011          4126  5                                    .DSTPTR [DST$L_BLI_BLKVEC_UNITS];
;  4012          4127  5                    IF .DSTPTR [DST$L_BLI_BLKVEC_BLOCKS] EQL 0
;  4013          4128  5                    THEN
;  4014          4129  5                        SUBVECTOR [0, DBG$L_PNSUB_UBOUND] = 0
```

```
;  4015          4130  5                    ELSE
;  4016          4131  5                        SUBVECTOR [0, DBG$L_PNSUB_UBOUND] =
;  4017          4132  5                                  .DSTPTR [DST$C_BLI_BLKVEC_BLOCKS]-1;
;  4018          4133  5
;  4019          4134  5
;  4020          4135  5                    ! Fill in stride and length as four (we are always
;  4021          4136  5                    ! representing blocks as blocks of longwords).
;  4022          4137  5                    !
;  4023          4138  5                    SUBVECTOR [1, DBG$L_PNSUB_STRIDE] = 4;
;  4024          4139  5                    NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_L;
;  4025          4140  5                    NODEPTR [DBG$W_PNARR_LENGTH] = 4;
;  4026          4141  5
;  4027          4142  5
;  4028          4143  5                    ! The upper bound on the subscript depends on the
;  4029          4144  5                    ! number of units per block.
;  4030          4145  5                    ! We temporarily dummy this up as if stride were 4, for
;  4031          4146  5                    ! purposes of aggregate output.
;  4032          4147  5                    !
;  4033          4148  5                    STRIDE_SIZE = .DSTPTR [DST$B_BLI_BLKVEC_UNIT_SIZE];
;  4034          4149  5                    IF (.DSTPTR [DST$L_BLI_BLKVEC_UNITS] EQL 0) OR
;  4035          4150  5                       (.STRIDE_SIZE EQL 0)
;  4036          4151  6                    THEN
;  4037          4152  5                        SUBVECTOR [1, DBG$L_PNSUB_UBOUND] = 0
;  4038          4153  5
;  4039          4154  5                    ELSE
;  4040          4155  5                        SUBVECTOR [1, DBG$L_PNSUB_UBOUND] =
;  4041          4156  5                          (.DSTPTR [DST$L_BLI_BLKVEC_UNITS]*.STRIDE_SIZE-1)
;  4042          4157  6                                                  /4;
;  4043          4158  5
;  4044          4159  5                    END;
;  4045          4160  4
;  4046          4161  4                        TES;
;  4047          4162  4
;  4048          4163  4                    END;
;  4049          4164  3
;  4050          4165  3
;  4051          4166  3                    ! For bitvectors, vectors, blocks, and blockvectors, and REF stuff,
;  4052          4167  3                    ! make sure the celltype is filled in correctly.
;  4053          4168  3                    !
;  4054          4169  3                    IF .FCODE EQL RST$K_TYPE_ARRAY
;  4055          4170  3                    THEN
;  4056          4171  3                        BEGIN
;  4057          4172  4                        ATOMIC_TYPE = .NODEPTR [DBG$B_PNARR_DTYPE];
;  4058          4173  4                        BIT_LENGTH = .NODEPTR [DBG$W_PNARR_LENGTH];
;  4059          4174  4                        IF (.ATOMIC_TYPE NEQ DSC$K_DTYPE_V)  AND
;  4060          4175  4                           (.ATOMIC_TYPE NEQ DSC$K_DTYPE_VU) AND
;  4061          4176  4                           (.ATOMIC_TYPE NEQ DSC$K_DTYPE_SV) AND
;  4062          4177  4                           (.ATOMIC_TYPE NEQ DSC$K_DTYPE_SVU)
;  4063          4178  5                        THEN
;  4064          4179  4                            BIT_LENGTH = .BIT_LENGTH * 8;
;  4065          4180  4
;  4066          4181  4                        NODEPTR [DBG$L_PNARR_CELLTYPE] =
;  4067          4182  4                                DBG$TYPEID_FOR_ATOMIC(.ATOMIC_TYPE, .BIT_LENGTH, FALSE);
;  4068          4183  4
;  4069          4184  4                        END;
;  4070          4185  3
;  4071          4186  3
```

```
;  4072          4187  3              IF .FCODE EQL RST$K_TYPE_PTR
;  4073          4188  3              THEN
;  4074          4189  4                  BEGIN
;  4075          4190  4                  ATOMIC_TYPE = DSC$K_DTYPE_L;
;  4076          4191  4                  BIT_LENGTH = 32;
;  4077          4192  4                  TYPEID = DBG$TYPEID_FOR_ATOMIC(.ATOMIC_TYPE, .BIT_LENGTH, FALSE);
;  4078          4193  3                  END;
;  4079          4194  3
;  4080          4195  2              END;
;  4081          4196  2
;  4082          4197  2
;  4083          4198  2          ! For all other cases, just allocate a Normal Sub-Node.  Also
;  4084          4199  2          ! clear the aggregate flag.
;  4085          4200  2          !
;  4086          4201  2          [INRANGE, OUTRANGE]:
;  4087          4202  3              BEGIN
;  4088          4203  3              NODEPTR = DBG$GET_TEMPMEM(DBG$K_PRIM_SIZE_NORMAL);
;  4089          4204  3              PRIMPTR[DBG$V_DHDR_AGGR] = FALSE;
;  4090          4205  2              END;
;  4091          4206  2
;  4092          4207  2          TES;
;  4093          4208  2
;  4094          4209  2
;  4095          4210  2      ! Fill in the standard fields common to all Primary Descriptor Sub-Nodes.
;  4096          4211  2      !
;  4097          4212  2      NODEPTR[DBG$B_PNODE_KIND] = .KIND;
;  4098          4213  2      NODEPTR[DBG$B_PNODE_FCODE] = .FCODE;
;  4099          4214  2      NODEPTR[DBG$L_PNODE_TYPEID] = .TYPEID;
;  4100          4215  2      NODEPTR[DBG$L_PNODE_SYMID] = .SYMID;
;  4101          4216  2      NODEPTR[DBG$L_PNODE_RELOC] = 0;
;  4102          4217  2
;  4103          4218  2
;  4104          4219  2      ! Also set the final Sub-Node's KIND, FCODE, and TYPEID in the Primary
;  4105          4220  2      ! Descriptor Root Node.  The Root Node thus describes the object described
;  4106          4221  2      ! by the Primary Symbol as a whole.
;  4107          4222  2      !
;  4108          4223  2      PRIMPTR[DBG$B_DHDR_KIND] = .KIND;
;  4109          4224  2      PRIMPTR[DBG$B_DHDR_FCODE] = .FCODE;
;  4110          4225  2      PRIMPTR[DBG$L_DHDR_TYPEID] = .TYPEID;
;  4111          4226  2
;  4112          4227  2
;  4113          4228  2      ! Append the Sub-Node to the Primary Descriptor by linking it in at the
;  4114          4229  2      ! end of the doubly linked Sub-Node chain.  Then return.
;  4115          4230  2      !
;  4116          4231  2      INSQUE(.NODEPTR,.PRIMPTR[DBG$L_PRIM_BLINK]);
;  4117          4232  2      RETURN;
;  4118          4233  2
;  4119          4234  1      END;


                                    0FFC 00000          .ENTRY  DBG$BUILD_PRIMARY_SUBNODE, Save R2,R3,R4,-  ; 3503
                                                                R5,R6,R7,R8,R9,R10,R11                          ;
                5E      FF74    CE  9E 00002            MOVAB   -140(SP), SP                                    ;
                58        04    AC  D0 00007            MOVL    PRIMPTR, R8                                     ; 3591
```

```
                00000000G  00          58  D0 0000B          MOVL    R8, DBG$GL_CURRENT_PRIMARY
                        15              01  10  AC  CF 00012          CASEL   FCODE, #1, #21                          ; 3598
                      002C            002C      003F      00017 1$:       .WORD   3$-1$,-                                 ;
                      002C            025B      002C      0001F           2$-1$,-                                 ;
                      002C            002C      002C      00027           2$-1$,-                                 ;
                      002C            002C      029F      0002F           2$-1$,-                                 ;
                      002C            0290      002C      00037           2$-1$,-                                 ;
                                        002C      002C      0003F           2$-1$,-                                 ;
                                                                          32$-1$,-                                ;
                                                                          2$-1$,-                                 ;
                                                                          2$-1$,-                                 ;
                                                                          2$-1$,-                                 ;
                                                                          2$-1$,-                                 ;
                                                                          2$-1$,-                                 ;
                                                                          36$-1$,-                                ;
                                                                          2$-1$,-                                 ;
                                                                          2$-1$,-                                 ;
                                                                          2$-1$,-                                 ;
                                                                          2$-1$,-                                 ;
                                                                          2$-1$,-                                 ;
                                                                          34$-1$,-                                ;
                                                                          2$-1$,-                                 ;
                                                                          2$-1$,-                                 ;
                                                                          2$-1$                                   ;
                            06  DD 00043 2$:      PUSHL   #6                                      ; 4203
                00000000G   00      01  FB 00045          CALLS   #1, DBG$GET_TEMPMEM                     ;
                            55      50  D0 0004C          MOVL    R0, NODEPTR                             ;
                        04  A8      01  8A 0004F          BICB2   #1, 4(R8)                               ; 4204
                                  047B  31 00053          BRW     66$                                     ; 3598
                            24  AE  9F 00056 3$:          PUSHAB  BITSIZE                                 ; 3615
                            04  AE  9F 00059              PUSHAB  DIMVECPTR                               ;
                            0C  AE  9F 0005C              PUSHAB  NDIMS                                   ;
                            14  AE  9F 0005F              PUSHAB  CELLTYPE                                ;
                            1C  AE  9F 00062              PUSHAB  DSCADDR                                 ;
                            14  AC  DD 00065              PUSHL   TYPEID                                  ;
                00000000G   00      06  FB 00068          CALLS   #6, DBG$STA_TYP_ARRAY                   ;
                            52  0C  AC  D0 0006F          MOVL    SYMID, R2                               ; 3631
                                    38  13 00073          BEQL    6$                                      ;
                            50      52  D0 00075          MOVL    R2, RSTPTR                              ; 3638
                            01  14  A0  91 00078 4$:      CMPB    20(RSTPTR), #1                          ; 3639
                                    06  13 0007C          BEQL    5$                                      ;
                            50  10  A0  D0 0007E          MOVL    16(RSTPTR), RSTPTR                      ; 3640
                                    F4  11 00082          BRB     4$                                      ;
                            06  29  A0  91 00084 5$:      CMPB    41(RSTPTR), #6                          ; 3641
                                    23  12 00088          BNEQ    6$                                      ;
                                    52  DD 0008A          PUSHL   R2                                      ; 3644
                00000000G   00      01  FB 0008C          CALLS   #1, DBG$STA_SETCONTEXT                  ;
                            10  AE  9F 00093              PUSHAB  VALUE_KIND                              ; 3645
                            34  AE  9F 00096              PUSHAB  DESC                                    ;
                                    52  DD 00099          PUSHL   R2                                      ;
                00000000G   00      03  FB 0009B          CALLS   #3, DBG$STA_SYMVALUE                    ;
                            03  10  AE  D1 000A2          CMPL    VALUE_KIND, #3                          ; 3646
                                    05  12 000A6          BNEQ    6$                                      ;
                        0C  AE  30  AE  D0 000A8          MOVL    DESC, DSCADDR                           ; 3648
                            59  04  AE  D0 000AD 6$:      MOVL    NDIMS, R9                               ; 3652
                            14          59  D1 000B1          CMPL    R9, #20                                 ;
                                        0D  15 000B4          BLEQ    7$                                      ;
```

```
                                00028EA8   8F   DD 000B6          PUSHL   #167592                                          ;
                    00000000G   00         01   FB 000BC          CALLS   #1, LIB$SIGNAL                                   ;
                            51             05   C5 000C3 7$:      MULL3   #5, R9, R1                                       ; 3654
                                       0A  A1   9F 000C7          PUSHAB  10(R1)                                           ; 3653
                    00000000G   00         01   FB 000CA          CALLS   #1, DBG$GET_TEMPMEM                              ;
                                55         50   D0 000D1          MOVL    R0, NODEPTR                                      ;
                                50     18  A5   9E 000D4          MOVAB   24(NODEPTR), R0                                  ; 3655
                            03  A0         59   90 000D8          MOVB    R9, 3(R0)                                        ;
                                52     0C  AE   D0 000DC          MOVL    DSCADDR, R2                                      ; 3663
                                0E     03  A2   91 000E0          CMPB    3(R2), #14                                       ;
                                           06   12 000E4          BNEQ    8$                                               ;
                            02  A0         22   90 000E6          MOVB    #34, 2(R0)                                       ; 3665
                                           05   11 000EA          BRB     9$                                               ;
                            02  A0     02  A2   90 000EC 8$:      MOVB    2(R2), 2(R0)                                     ; 3670
                            1C  A5         62   B0 000F1 9$:      MOVW    (R2), 28(NODEPTR)                                ; 3672
                                60     08  A2   B0 000F5          MOVW    8(R2), (R0)                                      ; 3673
                50      0A  A2             01   05   EF 000F9     EXTZV   #5, #1, 10(R2), R0                               ; 3675
            0A  A5          01             01   50   F0 000FF     INSV    R0, #1, #1, 10(NODEPTR)                          ;
                            24  A5     08  AE   D0 00105          MOVL    CELLTYPE, 36(NODEPTR)                            ; 3676
                                53     14  A2   9E 0010A          MOVAB   20(R2), STRIDEPTR                                ; 3682
                                5A       6349  DE 0010E          MOVAL   (STRIDEPTR)[R9], BOUNDVEC                        ; 3683
                        0D              01     03  A2   8F 00112 CASEB   3(R2), #1, #13                                  ; 3690
                002B            001C           001C     001C     00117 10$:  .WORD   11$-10$,-                            ;
                001C            001C           001C     001C     0011F       11$-10$,-                                    ;
                009C            001C           009C     001C     00127       11$-10$,-                                    ;
                                               0098     001C     0012F       12$-10$,-                                    ;
                                                                                11$-10$,-                                    ;
                                                                                11$-10$,-                                    ;
                                                                                11$-10$,-                                    ;
                                                                                11$-10$,-                                    ;
                                                                                11$-10$,-                                    ;
                                                                                23$-10$,-                                    ;
                                                                                11$-10$,-                                    ;
                                                                                23$-10$,-                                    ;
                                                                                11$-10$,-                                    ;
                                                                                23$-10$,-                                    ;
                                                                                11$-10$,-                                    ;
                                                                                22$-10$                                      ;
                                00028198   8F   DD 00133 11$:      PUSHL   #164248                                          ; 3781
                    00000000G   00         01   FB 00139          CALLS   #1, LIB$SIGNAL                                   ;
                                           71   11 00140          BRB     23$                                              ;
                                50     0C  A8   D0 00142 12$:      MOVL    12(R8), TMP_SYMID                                ; 3709
                                           17   13 00146          BEQL    15$                                              ; 3710
                                01     14  A0   91 00148 13$:      CMPB    20(TMP_SYMID), #1                                ; 3713
                                           06   13 0014C          BEQL    14$                                              ;
                                50     10  A0   D0 0014E          MOVL    16(TMP_SYMID), TMP_SYMID                         ; 3714
                                           F4   11 00152          BRB     13$                                              ;
                                05     29  A0   91 00154 14$:      CMPB    41(TMP_SYMID), #5                                ; 3715
                                           05   12 00158          BNEQ    15$                                              ;
            54          28  A0             05   E0 0015A          BBS     #5, 40(TMP_SYMID), 23$                           ; 3716
                                           52   DD 0015F 15$:      PUSHL   R2                                               ; 3725
                    00000000G   00         01   FB 00161          CALLS   #1, DBG$DATA_LENGTH                              ;
                                51     07  A0   9E 00168          MOVAB   7(R0), R1                                        ; 3726
                50              51             08   C7 0016C     DIVL3   #8, R1, LENGTH                                   ;
                                51     FF  A9   9E 00170          MOVAB   -1(R9), R1                                       ; 3736
                18      0A      A2             05   E1 00174     BBC     #5, 10(R2), 18$                                  ; 3732
                        3C      AE             50   D0 00179     MOVL    LENGTH, STRIDE                                   ; 3735
                                               50   D4 0017D     CLRL    I                                                ; 3736
                                               0A   11 0017F     BRB     17$                                              ;
```

```
                3C AE40      38 AE40      FC A340   C5 00181 16$:    MULL3   -4(STRIDEPTR)[I], STRIDE-4[I], STRIDE[I]    ; 3737
                     F2           50           51   F3 00188 17$:    AOBLEQ  R1, I, 16$                                  ;
                                  18           11   00018F           BRB     21$                                         ; 3732
                             3C AE41           50   D0 00191 18$:    MOVL    LENGTH, STRIDE[R1]                          ; 3747
                                  50       FF   A9   9E 00196        MOVAB   -1(R9), I                                   ; 3748
                                           0A   11   0019A           BRB     20$                                         ;
                3C AE40      40 AE40      04 A340   C5 0019C 19$:    MULL3   4(STRIDEPTR)[I], STRIDE+4[I], STRIDE[I]     ; 3749
                                  F3           50   F4 001A6 20$:    SOBGEQ  I, 19$                                      ;
                                  53       3C   AE   9E 001A9 21$:    MOVAB   STRIDE, STRIDEPTR                           ; 3756
                                           04   11   001AD           BRB     23$                                         ; 3690
                             0A   A5           04   88 001AF 22$:    BISB2   #4, 10(NODEPTR)                             ; 3774
                                  54       28   A5   9E 001B3 23$:    MOVAB   40(R5), SUBVECTOR                           ; 3793
                                           5B   D4   001B7           CLRL    OFFSET                                      ; 3794
                                  52           01   CE 001B9        MNEGL   #1, I                                       ; 3803
                                           009D 31   001BC           BRW     29$                                         ;
                                57           52   14   C5 001BF 24$:    MULL3   #20, I, R7                              ; 3797
                                56           52   01   78 001C3        ASHL    #1, I, R6                                   ;
                                           6744 9F   001C7           PUSHAB  (R7)[SUBVECTOR]                             ;
                                  9E       6A46 D0   001CA           MOVL    (BOUNDVEC)[R6], @(SP)+                      ;
                                           04 A744 9F 001CE          PUSHAB  4(R7)[SUBVECTOR]                            ; 3798
                                  9E       6342 D0   001D2           MOVL    (STRIDEPTR)[I], @(SP)+                      ;
                                           08 A744 9F 001D6          PUSHAB  8(R7)[SUBVECTOR]                            ; 3799
                                  9E       6A46 D0   001DA           MOVL    (BOUNDVEC)[R6], @(SP)+                      ;
                                           0C A744 9F 001DE          PUSHAB  12(R7)[SUBVECTOR]                           ; 3800
                                  9E       04 AA46 D0 001E2          MOVL    4(BOUNDVEC)[R6], @(SP)+                     ;
                                           20   AE   D4 001E7        CLRL    SUB_TYPEID                                  ; 3802
                                  50       00 BE42   D0 001EA        MOVL    @DIMVECPTR[I], R0                           ; 3803
                                           2F   13   001EF           BEQL    27$                                         ;
                                20   AE           50   D0 001F1        MOVL    R0, SUB_TYPEID                              ; 3806
                                  50       20   AE   D0 001F5 25$:    MOVL    SUB_TYPEID, R0                              ; 3807
                                  09       18   A0   91 001F9        CMPB    24(R0), #9                                  ;
                                           17   12   001FD           BNEQ    26$                                         ;
                                           14   AE   9F 001FF        PUSHAB  SIZE                                        ; 3808
                                           1C   AE   9F 00202        PUSHAB  HIGHPTR                                     ;
                                           24   AE   9F 00205        PUSHAB  LOWPTR                                      ;
                                           2C   AE   9F 00208        PUSHAB  SUB_TYPEID                                  ;
                                                50   DD 0020B        PUSHL   R0                                          ;
                00000000G    00                05   FB 0020D        CALLS   #5, DBG$STA_TYP_SUBRNG                      ;
                                                DF   11   00214      BRB     25$                                         ;
                                           10 A744 9F 00216 26$:    PUSHAB  16(R7)[SUBVECTOR]                           ; 3810
                                  9E       24   AE   D0 0021A        MOVL    SUB_TYPEID, @(SP)+                          ;
                                           07   11   0021E           BRB     28$                                         ; 3803
                                           10 A744 9F 00220 27$:    PUSHAB  16(R7)[SUBVECTOR]                           ; 3814
                                  9E           50   D0 00224        MOVL    R0, @(SP)+                                  ;
                                           04 A744 9F 00227 28$:    PUSHAB  4(R7)[SUBVECTOR]                            ; 3817
                           56    9E       6A46 C5   0022B          MULL3   (BOUNDVEC)[R6], @(SP)+, R6                  ;
                                  5B           56   C2 00230        SUBL2   R6, OFFSET                                  ; 3816
                                  09 00000000G 00   91 00233        CMPB    DBG$GB_LANGUAGE, #9                         ; 3825
                                           20   12   0023A           BNEQ    29$                                         ;
                                  50       20   AE   D0 0023C        MOVL    SUB_TYPEID, R0                              ; 3827
                                           1A   13   00240           BEQL    29$                                         ;
                                  04       18   A0   91 00242        CMPB    24(R0), #4                                  ; 3829
                                           14   12   00246           BNEQ    29$                                         ;
                                           6744 9F   00248           PUSHAB  (R7)[SUBVECTOR]                             ; 3833
                                                9E   DD 0024B        PUSHL   @(SP)+                                      ;
                                                50   DD 0024D        PUSHL   R0                                          ; 3832
                00000000G    00                02   FB 0024F        CALLS   #2, DBG$ENUM_VAL                            ;
```

```
                        6744  9F 00256          PUSHAB  (R7)[SUBVECTOR]                                    ;
                   9E     50  D0 00259          MOVL    R0, @(SP)+                                        ;
              02   52     59  F2 0025C 29$:     AOBLSS  R9, I, 30$                                        ; 3795
                          03  11 00260          BRB     31$                                               ;
                        FF5A  31 00262 30$:     BRW     24$                                               ;
              20   A5     5B  D0 00265 31$:     MOVL    OFFSET, 32(NODEPTR)                               ; 3841
                 43 00000000G  00  E9 00269     BLBC    DBG$GL_ARRSUB_FLAG, 35$                           ; 3842
                          2F  11 00270          BRB     33$                                               ; 3844
                     24   AE  9F 00272 32$:     PUSHAB  BITSIZE                                           ; 3856
                     2C   AE  9F 00275          PUSHAB  COMP_VEC                                          ;
                     34   AE  9F 00278          PUSHAB  NCOMPS                                            ;
                     14   AC  DD 0027B          PUSHL   TYPEID                                            ;
          00000000G  00      04  FB 0027E      CALLS   #4, DBG$STA_TYP_RECORD                            ;
                             07  DD 00285      PUSHL   #7                                                ; 3857
          00000000G  00      01  FB 00287      CALLS   #1, DBG$GET_TEMPMEM                               ;
                     55      50  D0 0028E      MOVL    R0, NODEPTR                                       ;
              18   A5         01  B0 00291      MOVW    #1, 24(NODEPTR)                                   ; 3858
              1A   A5     2C  AE  B0 00295      MOVW    NCOMPS, 26(NODEPTR)                               ; 3859
                 12 00000000G  00  E9 0029A     BLBC    DBG$GL_RECCMP_FLAG, 35$                           ; 3860
              04   A8         01  88 002A1 33$:     BISB2   #1, 4(R8)                                     ; 3862
                              0C  11 002A5          BRB     35$                                           ; 3598
                              0A  DD 002A7 34$:     PUSHL   #10                                           ; 3871
          00000000G  00      01  FB 002A9      CALLS   #1, DBG$GET_TEMPMEM                               ;
                     55      50  D0 002B0      MOVL    R0, NODEPTR                                       ;
                           021B  31 002B3 35$:     BRW     66$                                           ;
                     53   04   A8  9E 002B6 36$:     MOVAB   4(R8), R3                                     ; 3885
                     63         01  88 002BA          BISB2   #1, (R3)                                      ;
                           18   AC  D5 002BD          TSTL    DSTPTR                                        ; 3890
                              0E  12 002C0          BNEQ    37$                                           ;
                           0C   AC  D5 002C2          TSTL    SYMID                                         ;
                              09  13 002C5          BEQL    37$                                           ;
                     50    0C   AC  D0 002C7          MOVL    SYMID, R0                                     ; 3892
                  18   AC    0C   A0  D0 002CB          MOVL    12(R0), DSTPTR                                ;
                     52    18   AC  D0 002D0 37$:     MOVL    DSTPTR, R2                                    ; 3900
                           05   A2  95 002D4          TSTB    5(R2)                                         ;
                              1A  18 002D7          BGEQ    38$                                           ;
                           0C   AC  D5 002D9          TSTL    SYMID                                         ;
                              15  13 002DC          BEQL    38$                                           ;
                                  06  DD 002DE      PUSHL   #6                                            ; 3903
          00000000G  00           01  FB 002E0      CALLS   #1, DBG$GET_TEMPMEM                           ;
                     55           50  D0 002E7      MOVL    R0, NODEPTR                                   ;
                     63           01  8A 002EA      BICB2   #1, (R3)                                      ; 3904
                  10   AC         10  D0 002ED      MOVL    #16, FCODE                                    ; 3905
                                  23  11 002F1      BRB     41$                                           ; 3906
     56      05   A2              03  00  EF 002F3 38$:     EXTZV   #0, #3, 5(R2), R6                     ; 3914
                 04               00  56  CF 002F9          CASEL   R6, #0, #4                            ;
   00DD        00AC          001C        000A      002FD 39$:     .WORD   40$-39$,-                             ;
                                         0122      00305                 42$-39$,-                       ;
                                                                         52$-39$,-                       ;
                                                                         54$-39$,-                       ;
                                                                         58$-39$                         ;
                     63           01  8A 00307 40$:     BICB2   #1, (R3)                                  ; 3925
                                  06  DD 0030A          PUSHL   #6                                        ; 3926
          00000000G  00           01  FB 0030C      CALLS   #1, DBG$GET_TEMPMEM                           ;
                     55           50  D0 00313      MOVL    R0, NODEPTR                                   ;
                                 0169  31 00316 41$:     BRW     63$                                       ; 3914
                  10   AC         01  D0 00319 42$:     MOVL    #1, FCODE                                 ; 3936
```

```
                                0F  DD 0031D          PUSHL   #15                                  ; 3937
          00000000G  00         01  FB 0031F          CALLS   #1, DBG$GET_TEMPMEM                   ;
                     55         50  D0 00326          MOVL    R0, NODEPTR                          ;
                     51     18  A5  9E 00329          MOVAB   24(NODEPTR), R1                      ; 3939
               03    A1         01  90 0032D          MOVB    #1, 3(R1)                            ;
                     50     0A  A2  9E 00331          MOVAB   10(R2), R0                           ; 3940
  53          60     04         00  EF 00335          EXTZV   #0, #4, (R0), R3                     ;
               1C    A5         53  B0 0033A          MOVW    R3, 28(NODEPTR)                      ;
  01          60     04         00  ED 0033E          CMPZV   #0, #4, (R0), #1                     ; 3949
                                12  12 00343          BNEQ    44$                                  ;
               F0    8F         60  93 00345          BITB    (R0), #240                           ; 3951
                                06  13 00349          BEQL    43$                                  ;
               02    A1         06  90 0034B          MOVB    #6, 2(R1)                            ; 3953
                                4C  11 0034F          BRB     51$                                  ;
               02    A1         02  90 00351 43$:     MOVB    #2, 2(R1)                            ; 3955
                                46  11 00355          BRB     51$                                  ; 3951
  02          60     04         00  ED 00357 44$:     CMPZV   #0, #4, (R0), #2                     ; 3959
                                12  12 0035C          BNEQ    46$                                  ;
               F0    8F         60  93 0035E          BITB    (R0), #240                           ; 3961
                                06  13 00362          BEQL    45$                                  ;
               02    A1         07  90 00364          MOVB    #7, 2(R1)                            ; 3963
                                33  11 00368          BRB     51$                                  ;
               02    A1         03  90 0036A 45$:     MOVB    #3, 2(R1)                            ; 3965
                                2D  11 0036E          BRB     51$                                  ; 3961
  04          60     04         00  ED 00370 46$:     CMPZV   #0, #4, (R0), #4                     ; 3969
                                12  12 00375          BNEQ    48$                                  ;
               F0    8F         60  93 00377          BITB    (R0), #240                           ; 3971
                                06  13 0037B          BEQL    47$                                  ;
               02    A1         08  90 0037D          MOVB    #8, 2(R1)                            ; 3973
                                1A  11 00381          BRB     51$                                  ;
               02    A1         04  90 00383 47$:     MOVB    #4, 2(R1)                            ; 3975
                                14  11 00387          BRB     51$                                  ; 3971
               F0    8F         60  93 00389 48$:     BITB    (R0), #240                           ; 3981
                                06  13 0038D          BEQL    49$                                  ;
               02    A1         01  90 0038F          MOVB    #1, 2(R1)                            ; 3983
                                04  11 00393          BRB     50$                                  ;
               02    A1         22  90 00395 49$:     MOVB    #34, 2(R1)                           ; 3985
               1C    A5         08  A4 00399 50$:     MULW2   #8, 28(NODEPTR)                      ; 3992
                     54     28  A5  9E 0039D 51$:     MOVAB   40(R5), SUBVECTOR                    ; 3995
04  A4        60     04         00  EF 003A1          EXTZV   #0, #4, (R0), 4(SUBVECTOR)           ; 4003
                                24  11 003A7          BRB     53$                                  ; 4010
               10    AC         01  D0 003A9 52$:     MOVL    #1, FCODE                            ; 4027
                                0F  DD 003AD          PUSHL   #15                                  ; 4028
          00000000G  00         01  FB 003AF          CALLS   #1, DBG$GET_TEMPMEM                   ;
                     55         50  D0 003B6          MOVL    R0, NODEPTR                          ;
               0A    A5         04  88 003B9          BISB2   #4, 10(NODEPTR)                      ; 4034
               1A    A5 00010122 8F D0 003BD          MOVL    #65826, 26(NODEPTR)                  ; 4036
                     54     28  A5  9E 003C5          MOVAB   40(R5), SUBVECTOR                    ; 4038
               04    A4         01  D0 003C9          MOVL    #1, 4(SUBVECTOR)                     ; 4039
                            06  A2  D5 003CD 53$:     TSTL    6(R2)                                ; 4046
                                3A  13 003D0          BEQL    55$                                  ;
        0C    A4     06  A2     01  C3 003D2          SUBL3   #1, 6(R2), 12(SUBVECTOR)             ; 4052
                                43  11 003D8          BRB     57$                                  ; 3914
                     63         10  88 003DA 54$:     BISB2   #16, (R3)                            ; 4063
               10    AC         01  D0 003DD          MOVL    #1, FCODE                            ; 4064
                                0F  DD 003E1          PUSHL   #15                                  ; 4065
          00000000G  00         01  FB 003E3          CALLS   #1, DBG$GET_TEMPMEM                   ;
```

```
                                55              50  DO 003EA          MOVL    R0, NODEPTR                                     ;
                                54        28    A5  9E 003ED          MOVAB   40(R5), SUBVECTOR                       ; 4068
                          04    A4              04  DO 003F1          MOVL    #4, 4(SUBVECTOR)                        ; 4078
                          1A    A5  00040108    8F  DO 003F5          MOVL    #262408, 26(NODEPTR)                    ; 4079
          50      0A    A2              04      00  EF 003FD          EXTZV   #0, #4, 10(R2), STRIDE_SIZE             ; 4089
                                              06  A2  D5 00403          TSTL    6(R2)                                   ; 4090
                                                  04  13 00406          BEQL    55$
                                                  50  D5 00408          TSTL    STRIDE_SIZE                             ; 4091
                                                  05  12 0040A          BNEQ    56$
                                              0C  A4  D4 0040C  55$:     CLRL    12(SUBVECTOR)                           ; 4093
                                                  71  11 0040F          BRB     63$
                    51        06    A2              50  C5 00411  56$:     MULL3   STRIDE_SIZE, 6(R2), R1                  ; 4097
                                                  51  D7 00416          DECL    R1
                0C  A4              51              04  C7 00418          DIVL3   #4, R1, 12(SUBVECTOR)                   ; 4098
                                                  63  11 0041D  57$:     BRB     63$                                     ; 3914
                                63                10  88 0041F  58$:     BISB2   #16, (R3)                               ; 4109
                          10    AC                01  DO 00422          MOVL    #1, FCODE                               ; 4110
                                                  14  DD 00426          PUSHL   #20                                     ; 4111
              00000000G   00                      01  FB 00428          CALLS   #1, DBG$GET_TEMPMEM
                                55                50  DO 0042F          MOVL    R0, NODEPTR
                          1A    A5  00200201    8F  DO 00432          MOVL    #2097665, 26(NODEPTR)                   ; 4114
                                54        28    A5  9E 0043A          MOVAB   40(R5), SUBVECTOR                       ; 4116
                                51        0E    A2  9A 0043E          MOVZBL  14(R2), R1                              ; 4126
                0C  A4              51        0A    A2  C5 00442          MULL3   10(R2), R1, 4(SUBVECTOR)
                                              06  A2  D5 00448          TSTL    6(R2)                                   ; 4127
                                                  05  12 0044B          BNEQ    59$
                                              0C  A4  D4 0044D          CLRL    12(SUBVECTOR)                           ; 4129
                                                  06  11 00450          BRB     60$
                0C  A4        06    A2              01  C3 00452  59$:     SUBL3   #1, 6(R2), 12(SUBVECTOR)                ; 4133
                              18    A4              04  DO 00458  60$:     MOVL    #4, 24(SUBVECTOR)                       ; 4139
                              1A    A5              08  90 0045C          MOVB    #8, 26(NODEPTR)                         ; 4140
                              1C    A5              04  B0 00460          MOVW    #4, 28(NODEPTR)                         ; 4141
                                    50        0E    A2  9A 00464          MOVZBL  14(R2), STRIDE_SIZE                     ; 4149
                                              0A    A2  D5 00468          TSTL    10(R2)                                  ; 4150
                                                  04  13 0046B          BEQL    61$
                                                  50  D5 0046D          TSTL    STRIDE_SIZE                             ; 4151
                                                  05  12 0046F          BNEQ    62$
                                              20  A4  D4 00471  61$:     CLRL    32(SUBVECTOR)                           ; 4153
                                                  0C  11 00474          BRB     63$
                          52        0A    A2              50  C5 00476  62$:     MULL3   STRIDE_SIZE, 10(R2), R2         ; 4157
                                                  52  D7 0047B          DECL    R2
                20  A4              52              04  C7 0047D          DIVL3   #4, R2, 32(SUBVECTOR)                   ; 4158
                                    01        10    AC  D1 00482  63$:     CMPL    FCODE, #1                               ; 4170
                                                  2E  12 00486          BNEQ    65$
                                    52        1A    A5  9A 00488          MOVZBL  26(NODEPTR), ATOMIC_TYPE                ; 4173
                                    53        1C    A5  3C 0048C          MOVZWL  28(NODEPTR), BIT_LENGTH                 ; 4174
                                    01              52  D1 00490          CMPL    ATOMIC_TYPE, #1                         ; 4175
                                                  12  13 00493          BEQL    64$
                                    22              52  D1 00495          CMPL    ATOMIC_TYPE, #34                        ; 4176
                                                  0D  13 00498          BEQL    64$
                                    29              52  D1 0049A          CMPL    ATOMIC_TYPE, #41                        ; 4177
                                                  08  13 0049D          BEQL    64$
                                    2A              52  D1 0049F          CMPL    ATOMIC_TYPE, #42                        ; 4178
                                                  03  13 004A2          BEQL    64$
                                    53              08  C4 004A4          MULL2   #8, BIT_LENGTH                          ; 4180
                                                  7E  D4 004A7  64$:     CLRL    -(SP)                                   ; 4183
                                                  0C  BB 004A9          PUSHR   #^M<R2,R3>                              ;
```

```
                00000000G  00          03  FB 004AB          CALLS   #3, DBG$TYPEID_FOR_ATOMIC                ;
                      24   A5          50  D0 004B2          MOVL    R0, 36(NODEPTR)                         ;
                           10     10    AC  D1 004B6 65$:     CMPL    FCODE, #16                              ; 4187
                                        15  12 004BA          BNEQ    66$                                     ;
                           52           08  D0 004BC          MOVL    #8, ATOMIC_TYPE                         ; 4190
                           53           20  D0 004BF          MOVL    #32, BIT_LENGTH                         ; 4191
                                        7E  D4 004C2          CLRL    -(SP)                                   ; 4192
                                        0C  BB 004C4          PUSHR   #^M<R2,R3>                              ;
                00000000G  00          03  FB 004C6          CALLS   #3, DBG$TYPEID_FOR_ATOMIC               ;
                      14   AC          50  D0 004CD          MOVL    R0, TYPEID                              ;
                      08   A5     08   AC  90 004D1 66$:     MOVB    KIND, 8(NODEPTR)                        ; 4212
                      09   A5     10   AC  90 004D6          MOVB    FCODE, 9(NODEPTR)                       ; 4213
                      0C   A5     14   AC  D0 004DB          MOVL    TYPEID, 12(NODEPTR)                     ; 4214
                      10   A5     0C   AC  D0 004E0          MOVL    SYMID, 16(NODEPTR)                      ; 4215
                                  14   A5  D4 004E5          CLRL    20(NODEPTR)                             ; 4216
                      07   A8     08   AC  90 004E8          MOVB    KIND, 7(R8)                             ; 4223
                      06   A8     10   AC  90 004ED          MOVB    FCODE, 6(R8)                            ; 4224
                      08   A8     14   AC  D0 004F2          MOVL    TYPEID, 8(R8)                           ; 4225
                      18   B8          65  0E 004F7          INSQUE  (NODEPTR), @24(R8)                      ; 4231
                                           04 004FB          RET                                             ; 4234

; Routine Size:  1276 bytes,    Routine Base:  DBG$CODE + 006C
```

```
;  4121          4235  1 GLOBAL ROUTINE DBG$EXP_INT(INPUT_DESC, RADIX, VALUE_PTR, TERM_INDEX) =
;  4122          4236  1 !
;  4123          4237  1 ! FUNCTION
;  4124          4238  1 !       This is the common Expression Interpreter for most languages
;  4125          4239  1 !       supported by DEBUG.  It parses and evaluates a source language
;  4126          4240  1 !       expression and returns a Value Descriptor which represents the
;  4127          4241  1 !       value of the expression.  This routine itself is only a set-up
;  4128          4242  1 !       routine which sets up the character pointer and the expression
;  4129          4243  1 !       radix to use and then calls DBG$EXPRESSION_PARSER to do the
;  4130          4244  1 !       actual work.
;  4131          4245  1 !
;  4132          4246  1 ! INPUTS
;  4133          4247  1 !       INPUT_DESC - The address of a VAX standard string descriptor which
;  4134          4248  1 !                    describes the input string to be parsed.  The length is
;  4135          4249  1 !                    actually not used, however--the string is instead assumed
;  4136          4250  1 !                    to be terminated by a carriage-return character.
;  4137          4251  1 !
;  4138          4252  1 !       RADIX    - The radix to be used to interpret integer constants in the
;  4139          4253  1 !                    input string.  The allowed radix values are DBG$K_DECIMAL,
;  4140          4254  1 !                    DBG$K_HEX, DBG$K_OCTAL, and DBG$K_BINARY.
;  4141          4255  1 !
;  4142          4256  1 !       VALUE_PTR - The address of a longword to receive a pointer to the
;  4143          4257  1 !                    value descriptor returned by this routine as its output.
;  4144          4258  1 !
;  4145          4259  1 !       TERM_INDEX - A "terminator index" which indicates which lexical tokens
;  4146          4260  1 !                    are allowed as expression terminators in this context.  These
;  4147          4261  1 !                    index values have names of the form TOKEN$K_TERM_xxx.
;  4148          4262  1 !
;  4149          4263  1 !       5th Optional Parameter - If this is present, and the value is
;  4150          4264  1 !                            DBG$K_DEPOSIT_VERB then pass this into
;  4151          4265  1 !                            DBG$EXPRESSION_PARSER, so that in
;  4152          4266  1 !                            DBG$EXPRESSION_PARSER, when the expression is not
;  4153          4267  1 !                            address expression and in deposit command,
;  4154          4268  1 !                            DBG$EVAL_LANG_OPERATOR will not be called with
;  4155          4269  1 !                            DBG$GL_IDENTITY_TOKEN.
;  4156          4270  1 !                            (This is passed in from DBG$NPARSE_DEPOSIT, from
;  4157          4271  1 !                             DBG$NPARSE_EXPRESSION).
;  4158          4272  1 !
;  4159          4273  1 ! OUTPUTS
;  4160          4274  1 !       VALUE_PTR - The address of a Value Descriptor is returned to VALUE_PTR.
;  4161          4275  1 !                   This Value Descriptor represents the value of the expression
;  4162          4276  1 !                   interpreted by DBG$EXP_INT.
;  4163          4277  1 !
;  4164          4278  1 !       The value STS$K_SUCCESS is return as the routine result if the expres-
;  4165          4279  1 !                   sion was terminated by a carriage-return character.  If it
;  4166          4280  1 !                   was terminated any other way (i.e., by a terminator token),
;  4167          4281  1 !                   the value STS$K_WARNING is returned.
;  4168          4282  1 !
;  4169          4283  1
;  4170          4284  2     BEGIN
;  4171          4285  2
;  4172          4286  2     MAP
;  4173          4287  2         INPUT_DESC: REF BLOCK[,BYTE],   ! Pointer to input string descriptor
;  4174          4288  2         VALUE_PTR: REF VECTOR[1];       ! Longword to receive Value Descr ptr.
;  4175          4289  2
;  4176          4290  2     LOCAL
;  4177          4291  2         VALPTR: REF DBG$VALDESC;        ! Pointer to returned Value Descriptor
```

```
; 4178          4292  2
; 4179          4293  2     BUILTIN ACTUALCOUNT,ACTUALPARAMETER;
; 4180          4294  2
; 4181          4295  2
; 4182          4296  2     ! Set up CHARPTR to point to the start of the expression string.  Also set
; 4183          4297  2     ! up the radix we are to use in the scan and initialize some variables.
; 4184          4298  2     !
; 4185          4299  2     CHARPTR = .INPUT_DESC[DSC$A_POINTER];
; 4186          4300  2     EXPRESSION_RADIX = .RADIX;
; 4187          4301  2     SAVED_TOKEN = 0;
; 4188          4302  2
; 4189          4303  2
; 4190          4304  2     ! Call the Expression Parser to parse the language expression.  If the
; 4191          4305  2     ! result is an unconverted constant Value Descriptor, we convert it to
; 4192          4306  2     ! a real Value Descriptor here.  Then return the Value Descriptor pointer.
; 4193          4307  2     !
; 4194          4308  2     IF ACTUALCOUNT() GTR 4
; 4195          4309  2     THEN
; 4196          4310  3         BEGIN
; 4197          4311  3         IF ACTUALPARAMETER(5) NEQ DBG$K_DEPOSIT_VERB
; 4198          4312  3         THEN
; 4199          4313  3             $DBG_ERROR('DBGPARSER\DBG$EXP_INT');
; 4200          4314  3
; 4201          4315  3         VALPTR = DBG$EXPRESSION_PARSER(FALSE,
; 4202          4316  3                                        .TERM_POINTER_TBL[.TERM_INDEX] + TABLEBASE,
; 4203          4317  3                                        DBG$K_DEPOSIT_VERB);
; 4204          4318  3         END
; 4205          4319  3
; 4206          4320  2     ELSE
; 4207          4321  2         VALPTR = DBG$EXPRESSION_PARSER(FALSE,
; 4208          4322  2                                        .TERM_POINTER_TBL[.TERM_INDEX] + TABLEBASE);
; 4209          4323  2     VALUE_PTR[0] = .VALPTR;
; 4210          4324  2
; 4211          4325  2
; 4212          4326  2     ! Fix up the string descriptor to reflect the new location of the parse
; 4213          4327  2     ! pointer and return the appropriate status code.
; 4214          4328  2     !
; 4215          4329  2     INPUT_DESC[DSC$W_LENGTH] = .INPUT_DESC[DSC$W_LENGTH] +
; 4216          4330  2                                .INPUT_DESC[DSC$A_POINTER] - .CHARPTR;
; 4217          4331  2     INPUT_DESC[DSC$A_POINTER] = .CHARPTR;
; 4218          4332  2     IF .CHARPTR[0] NEQ CAR_RET THEN RETURN STS$K_WARNING;
; 4219          4333  2     RETURN STS$K_SUCCESS;
; 4220          4334  2
; 4221          4335  1     END;


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

24 47 42 44 5C 52 45 53 52 41 50 47 42 44 15  03040 P.AWR:    .ASCII  <21>\DBGPARSER\<92>\DBG$EXP_INT\        ;
                        54 4E 49 5F 50 58 45  0304F                                                     ;


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

                                              003C 00000        .ENTRY  DBG$EXP_INT, Save R2,R3,R4,R5           ; 4235
```

```
                55 00000000'   EF  9E 00002          MOVAB   CHARPTR, R5                                ;
                54 00000000'   EF  9E 00009          MOVAB   TERM_POINTER_TBL, R4                      ;
                53       04    AC  D0 00010          MOVL    INPUT_DESC, R3                            ; 4299
                65       04    A3  D0 00014          MOVL    4(R3), CHARPTR                            ;
           040C C5       08    AC  D0 00018          MOVL    RADIX, EXPRESSION_RADIX                   ; 4300
                      0424    C5  D4 0001E          CLRL    SAVED_TOKEN                               ; 4301
                52       10    AC  D0 00022          MOVL    TERM_INDEX, R2                            ; 4316
                04             6C  91 00026          CMPB    (AP), #4                                  ; 4308
                               30  1B 00029          BLEQU   2$                                        ;
                05       14    AC  D1 0002B          CMPL    20(AP), #5                                ; 4311
                               13  13 0002F          BEQL    1$                                        ;
                            2A6C   C4  9F 00031      PUSHAB  P.AWR                                     ; 4313
                               01  DD 00035          PUSHL   #1                                        ;
                      00028362     8F  DD 00037      PUSHL   #164706                                   ;
00000000G       00             03  FB 0003D          CALLS   #3, LIB$SIGNAL                            ;
                               05  DD 00044 1$:      PUSHL   #5                                        ; 4315
                50          6442  D0 00046          MOVL    TERM_POINTER_TBL[R2], R0                  ; 4316
                51       FAAF   C4  9E 0004A          MOVAB   TABLEBASE, R1                             ;
                            6140  9F 0004F          PUSHAB  (R1)[R0]                                  ;
                               7E  D4 00052          CLRL    -(SP)                                     ; 4315
            0000V CF           03  FB 00054          CALLS   #3, DBG$EXPRESSION_PARSER                 ;
                               13  11 00059          BRB     3$                                        ; 4308
                50          6442  D0 0005B 2$:      MOVL    TERM_POINTER_TBL[R2], R0                  ; 4322
                51       FAAF   C4  9E 0005F          MOVAB   TABLEBASE, R1                             ;
                            6140  9F 00064          PUSHAB  (R1)[R0]                                  ;
                               7E  D4 00067          CLRL    -(SP)                                     ; 4321
            0000V CF           02  FB 00069          CALLS   #2, DBG$EXPRESSION_PARSER                 ;
              0C  BC           50  D0 0006E 3$:      MOVL    VALPTR, @VALUE_PTR                        ; 4323
                51             63  3C 00072          MOVZWL  (R3), R1                                  ; 4330
                51       04    A3  C0 00075          ADDL2   4(R3), R1                                 ;
                50             65  D0 00079          MOVL    CHARPTR, R0                               ;
                   63          51  50 A3 0007C       SUBW3   R0, R1, (R3)                              ;
                      04       A3  50 D0 00080       MOVL    R0, 4(R3)                                 ; 4331
                               0D  60 91 00084       CMPB    (R0), #13                                 ; 4332
                               04  12 00087          BNEQ    4$                                        ;
                50             01  D0 00089          MOVL    #1, R0                                    ; 4333
                                   04 0008C          RET                                               ;
                               50  D4 0008D 4$:      CLRL    R0                                        ; 4335
                                   04 0008F          RET                                               ;

; Routine Size:  144 bytes,    Routine Base:  DBG$CODE + 0568
```

```
;4223     4336  1 GLOBAL ROUTINE DBG$EXPRESSION_PARSER(ADDRESS_EXPRESSION, TERM_LIST) =
;4224     4337  1 !
;4225     4338  1 ! FUNCTION
;4226     4339  1 !       This routine parses and interprets either a DEBUG Address Expression or
;4227     4340  1 !       a language expression in the current language and returns the result of
;4228     4341  1 !       the expression evaluation.
;4229     4342  1 !
;4230     4343  1 !       The routine uses an Operator Precedence parsing scheme.  Each operator
;4231     4344  1 !       is represented by an Operator Lexical Token Entry which contains the
;4232     4345  1 !       kind of the operator (prefix, infix, or postfix) and the left and right
;4233     4346  1 !       precedences of that operator.  Operands are represented by Primary
;4234     4347  1 !       Descriptors, Value Descriptors, or other Operand Lexical Token Entries.
;4235     4348  1 !       The operators and operands are retrieved by calling the Primary Parser.
;4236     4349  1 !
;4237     4350  1 !       When an operand is encountered, it is simply stacked on the operand
;4238     4351  1 !       stack.  When an operator is encountered, its left precedence is com-
;4239     4352  1 !       pared to the right precedence of the previous operator on the operator
;4240     4353  1 !       stack.  If the previous operator has the higher or equal precedence,
;4241     4354  1 !       it is popped from the operator stack and evaluated.  The evaluation
;4242     4355  1 !       requires one or two operands to be popped from the operand stack, after
;4243     4356  1 !       which the result is pushed back on that stack.  When no previous opera-
;4244     4357  1 !       tor has a higher or equal precedence, the new operator is pushed onto
;4245     4358  1 !       the operator stack.
;4246     4359  1 !
;4247     4360  1 !       The operator stack is always initialized with the "initiator operator"
;4248     4361  1 !       which ensures that there is always a previous operator on the stack.
;4249     4362  1 !       The end of an expression is always signalled by the "terminator opera-
;4250     4363  1 !       tor" whose left precedence is set such that it forces evaluation of all
;4251     4364  1 !       operators still on the operator stack up to the initiator operator.
;4252     4365  1 !       The single operand left on the operand stack thereafter constitutes
;4253     4366  1 !       the result of the expression evaluation.
;4254     4367  1 !
;4255     4368  1 !       This routine accepts a list of allowed "terminator tokens" (keywords
;4256     4369  1 !       such as "DO" or "THEN" or special characters such as "(", ")", or "=",
;4257     4370  1 !       depending on context).  This list is passed to the Lexical Scanner
;4258     4371  1 !       which returns the Terminator Operator when such a token or a carriage-
;4259     4372  1 !       return is encountered.  As a side effect, OWN variable TERMINATOR_CODE
;4260     4373  1 !       is set to a value which indicates which terminator token was found.
;4261     4374  1 !       That terminator's character length is also set in TERMINATOR_LENGTH.
;4262     4375  1 !       (This side effect is used when parsing subscript expressions.)
;4263     4376  1 !
;4264     4377  1 ! INPUTS
;4265     4378  1 !       ADDRESS_EXPRESSION - A flag set to TRUE if a DEBUG Address Expression
;4266     4379  1 !                   is to be parsed and evaluated.  If this flag is FALSE, a
;4267     4380  1 !                   language expression for the current language is parsed and
;4268     4381  1 !                   evaluated instead.  This flag affects both the lexical
;4269     4382  1 !                   scanning of operator symbols and the parsing and evaluation
;4270     4383  1 !                   of the expression operators.
;4271     4384  1 !
;4272     4385  1 !       TERM_LIST - A vector of pointers to Terminator Lexical Token Entries
;4273     4386  1 !                   for the Terminator Tokens which can terminate the expression
;4274     4387  1 !                   to be parsed.  The vector must be in PLIT form (TERM_LIST[-1]
;4275     4388  1 !                   gives the number of entries) and each pointer is expected to
;4276     4389  1 !                   be relative to TABLEBASE.  If there are no terminator tokens
;4277     4390  1 !                   other than carriage return, this list is empty (0 entries).
;4278     4391  1 !
;4279     4392  1 !       3rd Optional Parameter - If this is present, and the value is
```

```
;  4280          4393  1 !                       DBG$K_DEPOSIT_VERB then this is passed in from
;  4281          4394  1 !                       DBG$EXP_INT, from DBG$NPARSE_EXPRESSION, and from
;  4282          4395  1 !                       DBG$NPARSE_DEPSOIT, , so that in
;  4283          4396  1 !                       DBG$EXPRESSION_PARSER, when the expression is not
;  4284          4397  1 !                       address expression and in deposit command,
;  4285          4398  1 !                       DBG$EVAL_LANG_OPERATOR will not be called with
;  4286          4399  1 !                       DBG$GL_IDENTITY_TOKEN.
;  4287          4400  1 !
;  4288          4401  1 ! OUTPUTS
;  4289          4402  1 !       A pointer to a Primary Descriptor or a Value Descriptor specifying the
;  4290          4403  1 !                 result of the expression evaluation is returned as the
;  4291          4404  1 !                 routine value.
;  4292          4405  1 !
;  4293          4406  1
;  4294          4407  2     BEGIN
;  4295          4408  2
;  4296          4409  2     MAP
;  4297          4410  2         TERM_LIST: REF VECTOR[,LONG];    ! Pointer to Terminator Table
;  4298          4411  2
;  4299          4412  2     LITERAL
;  4300          4413  2         MAX_OPAND_INDEX = 25,            ! Maximum size of operand stack
;  4301          4414  2         MAX_OPTOR_INDEX = 25;            ! Maximum size of operator stack
;  4302          4415  2
;  4303          4416  2     LOCAL
;  4304          4417  2         DEPOSIT_FLAG,                    ! A flag to indicate the call is
;  4305          4418  2                                          !      from DBG$NPARSE_DEPOSIT
;  4306          4419  2         JUNK,                            ! Output parameter - not used
;  4307          4420  2         LEFT_ARG,                        ! Pointer to the left (or only) argument
;  4308          4421  2                                          !      of the current operator
;  4309          4422  2         LEFT_OP: REF TOKEN$ENTRY,        ! Pointer to top operator on operator
;  4310          4423  2                                          !      stack--operator to the left
;  4311          4424  2                                          !      of the current operator
;  4312          4425  2         NEW_PRIMPTR: REF DBG$PRIMARY,    ! Pointer to a Primary Descriptor
;  4313          4426  2         OP,                              ! The operator code for current operator
;  4314          4427  2         OPAND_INDEX,                     ! Current index into OPERAND_STACK
;  4315          4428  2         OPERAND_EXPECTED,                ! Flag set when operand or prefix ope-
;  4316          4429  2                                          !      rator is expected next
;  4317          4430  2         OPERAND_STACK:                   ! Parser's operand stack
;  4318          4431  2                 VECTOR[MAX_OPAND_INDEX],
;  4319          4432  2         OPERATOR_STACK:                  ! Parser's operator stack
;  4320          4433  2                 VECTOR[MAX_OPTOR_INDEX],
;  4321          4434  2         OPTOR_INDEX,                     ! Current index into OPERATOR_STACK
;  4322          4435  2         PAREN_NESTING,                   ! Parenthesis nesting depth in expr
;  4323          4436  2         PRIMPTR: REF DBG$PRIMARY,        ! Pointer to a Primary Descriptor
;  4324          4437  2         RIGHT_ARG,                       ! Pointer to the right argument to the
;  4325          4438  2                                          !      current operator
;  4326          4439  2         TEMP_TOKEN: REF TOKEN$ENTRY,     ! Temporary copy of bit-select token
;  4327          4440  2         TOKEN: REF TOKEN$ENTRY,          ! Pointer to the current Token Entry
;  4328          4441  2         TOKEN_OPERAND_FLAG,              ! Flag returned by Primary Parser
;  4329          4442  2                                          !      saying an operand was returned
;  4330          4443  2         VALPTR: REF DBG$PRIMARY;         ! Pointer to Primary or Value Descriptor
;  4331          4444  2
;  4332          4445  2
;  4333          4446  2     BUILTIN ACTUALCOUNT,ACTUALPARAMETER;
;  4334          4447  2
;  4335          4448  2
;  4336          4449  2     DEPOSIT_FLAG = FALSE;
```

```
;  4337          4450  2      IF ACTUALCOUNT() GTR 2
;  4338          4451  2      THEN
;  4339          4452  3          BEGIN
;  4340          4453  3          IF ACTUALPARAMETER(3) NEQ DBG$K_DEPOSIT_VERB
;  4341          4454  3          THEN
;  4342          4455  3              $DBG_ERROR('DBGNPARSE\DBG$NPARSE_EXPRESSION');
;  4343          4456  3
;  4344          4457  3          DEPOSIT_FLAG = TRUE;
;  4345          4458  2          END;
;  4346          4459  2
;  4347          4460  2
;  4348          4461  2      ! Initialize the operator stack to contain one operator, the initiator
;  4349          4462  2      ! operator.  Also initialize the operand stack to be empty.  Indicate
;  4350          4463  2      ! that we expect an operand at the start of the parse.
;  4351          4464  2      !
;  4352          4465  2      OPTOR_INDEX = 0;
;  4353          4466  2      OPERATOR_STACK[.OPTOR_INDEX] = INITIATOR_TOKEN;
;  4354          4467  2      OPAND_INDEX = -1;
;  4355          4468  2      OPERAND_EXPECTED = TRUE;
;  4356          4469  2      PAREN_NESTING = 0;
;  4357          4470  2
;  4358          4471  2
;  4359          4472  2      ! Loop through all operands and operators on the input line being parsed
;  4360          4473  2      ! until we reach the terminator operator.  Stack operands and stack or
;  4361          4474  2      ! evaluate operators as appropriate until the terminator operator forces
;  4362          4475  2      ! evaluation of all stacked operators.
;  4363          4476  2      !
;  4364          4477  2      WHILE TRUE DO
;  4365          4478  3          BEGIN
;  4366          4479  3
;  4367          4480  3
;  4368          4481  3          ! Pick up the next operand or operator.  An operand is returned as a
;  4369          4482  3          ! Primary Descriptor or a Value Descriptor by the Primary Parser.
;  4370          4483  3          ! An operator is just returned as an Operator Lexical Token Entry.
;  4371          4484  3          !
;  4372          4485  3          DBG$PRIMARY_PARSER(.OPERAND_EXPECTED, .ADDRESS_EXPRESSION,
;  4373          4486  3                             .TERM_LIST, .PAREN_NESTING, TOKEN, TOKEN_OPERAND_FLAG);
;  4374          4487  3
;  4375          4488  3
;  4376          4489  3          ! Handle operands.  If this is an operand, check that we are actually
;  4377          4490  3          ! expecting an operand at this point.  Then stack the operand on the
;  4378          4491  3          ! operand stack and loop to get the next operand or operator.
;  4379          4492  3          !
;  4380          4493  3          IF .TOKEN_OPERAND_FLAG
;  4381          4494  3          THEN
;  4382          4495  4              BEGIN
;  4383          4496  4              IF NOT .OPERAND_EXPECTED
;  4384          4497  4              THEN
;  4385          4498  4                  !<<----------- FIX UP MESSAGE ---------------------------------
;  4386          4499  4                  SIGNAL(DBG$_MISINVOPER, 1, UPLIT BYTE(%ASCIC 'somewhere'));
;  4387          4500  4
;  4388          4501  4              OPERAND_EXPECTED = FALSE;
;  4389          4502  4              OPAND_INDEX = .OPAND_INDEX + 1;
;  4390          4503  4              IF .OPAND_INDEX GEQ MAX_OPAND_INDEX THEN SIGNAL(DBG$_PARSTKOVR);
;  4391          4504  4              OPERAND_STACK[.OPAND_INDEX] = .TOKEN;
;  4392          4505  4              IF .DBG$GL_DEVELOPER[3] THEN DUMP_PRIMARY(.TOKEN);
;  4393          4506  4              END
```

```
;  4394          4507  4
;  4395          4508  4
;  4396          4509  4          ! Handle operators.  First check that an operator of this operator's
;  4397          4510  4          ! kind is expected.  Then loop to pop all higher-precedence operators
;  4398          4511  4          ! from the operator stack so they can be evaluated.  When no higher-
;  4399          4512  4          ! precedence operators are left on the stack, we stack the current
;  4400          4513  4          ! operator on the operator stack.
;  4401          4514  4          !
;  4402          4515  3          ELSE
;  4403          4516  4              BEGIN
;  4404          4517  4
;  4405          4518  4
;  4406          4519  4              ! Check that an operator was expected unless this is a prefix
;  4407          4520  4              ! operator (which is okay when we expect an operand).  This check
;  4408          4521  4              ! catches any expression which is not well-formed, including empty
;  4409          4522  4              ! expressions.  Also say that we expect an operand next unless
;  4410          4523  4              ! this is a postfix operator.
;  4411          4524  4              !
;  4412          4525  5              IF (.OPERAND_EXPECTED AND
;  4413          4526  4                      (.TOKEN[TOKEN$B_KIND] NEQ TOKEN$K_PREFIX_OP)) OR
;  4414          4527  5                 ((NOT .OPERAND_EXPECTED) AND
;  4415          4528  5                      (.TOKEN[TOKEN$B_KIND] EQL TOKEN$K_PREFIX_OP))
;  4416          4529  4              THEN
;  4417          4530  4                  SIGNAL(DBG$_MISOPEMIS, 1, TOKEN[TOKEN$B_OPLEN]);
;  4418          4531  4
;  4419          4532  4              IF .TOKEN[TOKEN$B_KIND] NEQ TOKEN$K_POSTFIX_OP
;  4420          4533  4              THEN
;  4421          4534  4                  OPERAND_EXPECTED = TRUE;
;  4422          4535  4
;  4423          4536  4
;  4424          4537  4              ! Pop and evaluate all operators on the operator stack whose prece-
;  4425          4538  4              ! dence is higher than or equal to the current operator's.  When we
;  4426          4539  4              ! finally reach a lower-precedence operator, we exit the pop loop.
;  4427          4540  4              !
;  4428          4541  4              WHILE TRUE DO
;  4429          4542  5                  BEGIN
;  4430          4543  5
;  4431          4544  5
;  4432          4545  5                  ! If the current operator (TOKEN) has a precedence higher than
;  4433          4546  5                  ! the precedence of the top operator on the stack (LEFT_OP),
;  4434          4547  5                  ! then leave this loop.
;  4435          4548  5                  !
;  4436          4549  5                  LEFT_OP = .OPERATOR_STACK[.OPTOR_INDEX];
;  4437          4550  5                  IF .LEFT_OP[TOKEN$B_R_PREC] LSS .TOKEN[TOKEN$B_L_PREC]
;  4438          4551  5                  THEN
;  4439          4552  5                      EXITLOOP;
;  4440          4553  5
;  4441          4554  5
;  4442          4555  5                  ! The LEFT_OP operator has a higher or equal precedence.  Pop
;  4443          4556  5                  ! it from the operator stack and evaluate it.  Start by picking
;  4444          4557  5                  ! up its operands from the operand stack.
;  4445          4558  5                  !
;  4446          4559  5                  OPTOR_INDEX = .OPTOR_INDEX - 1;
;  4447          4560  5                  IF .OPTOR_INDEX LSS 0
;  4448          4561  5                  THEN
;  4449          4562  5                      $DBG_ERROR('DBGPARSER\EXPRESSION_PARSER 1');
;  4450          4563  5
```

```
;  4451          4564  5          LEFT_ARG = .OPERAND_STACK[.OPAND_INDEX];
;  4452          4565  5          RIGHT_ARG = 0;
;  4453          4566  5          IF .LEFT_OP[TOKEN$B_KIND] EQL TOKEN$K_INFIX_OP
;  4454          4567  5          THEN
;  4455          4568  6              BEGIN
;  4456          4569  6              RIGHT_ARG = .LEFT_ARG;
;  4457          4570  6              OPAND_INDEX = .OPAND_INDEX - 1;
;  4458          4571  6              LEFT_ARG = .OPERAND_STACK[.OPAND_INDEX];
;  4459          4572  5              END;
;  4460          4573  5
;  4461          4574  5          IF .DBG$GL_DEVELOPER[3] THEN DUMP_OPERATOR(.LEFT_OP, FALSE);
;  4462          4575  5
;  4463          4576  5
;  4464          4577  5          ! If the operator we are about to evaluate is a "lexical"
;  4465          4578  5          ! operator, meaning that it has some semantic effect on the
;  4466          4579  5          ! lexical processing or parsing of the current expression,
;  4467          4580  5          ! we perform that semantic action here.
;  4468          4581  5          !
;  4469          4582  5          IF  .LEFT_OP[TOKEN$V_LEXICAL]
;  4470          4583  5          THEN
;  4471          4584  6              BEGIN
;  4472          4585  6              SELECTONE .LEFT_OP[TOKEN$W_CODE] OF
;  4473          4586  6                  SET
;  4474          4587  6
;  4475          4588  6
;  4476          4589  6                  ! If evaluation of an open parenthesis operator is
;  4477          4590  6                  ! forced by the terminator operator, we signal
;  4478          4591  6                  ! unbalanced parentheses.
;  4479          4592  6                  !
;  4480          4593  6                  [TOKEN$K_OPENPAREN]:
;  4481          4594  6                      SIGNAL(DBG$_UNBPAREN);
;  4482          4595  6
;  4483          4596  6
;  4484          4597  6                  ! Restore the current radix to decimal.  This undoes
;  4485          4598  6                  ! a previous radix operator (such as %HEX) in the input.
;  4486          4599  6                  !
;  4487          4600  6                  [TOKEN$K_RADIX_DEC]:
;  4488          4601  6                      EXPRESSION_RADIX = DBG$K_DECIMAL;
;  4489          4602  6
;  4490          4603  6
;  4491          4604  6                  ! Restore the current radix to hexadecimal.  This undoes
;  4492          4605  6                  ! a previous radix operator (such as %DEC) in the input.
;  4493          4606  6                  !
;  4494          4607  6                  [TOKEN$K_RADIX_HEX]:
;  4495          4608  6                      EXPRESSION_RADIX = DBG$K_HEX;
;  4496          4609  6
;  4497          4610  6
;  4498          4611  6                  ! Restore the current radix to octal.  This undoes a
;  4499          4612  6                  ! previous radix operator (such as %HEX) in the input.
;  4500          4613  6                  !
;  4501          4614  6                  [TOKEN$K_RADIX_OCT]:
;  4502          4615  6                      EXPRESSION_RADIX = DBG$K_OCTAL;
;  4503          4616  6
;  4504          4617  6
;  4505          4618  6                  ! Restore the current radix to binary.  This undoes a
;  4506          4619  6                  ! previous radix operator (such as %HEX) in the input.
;  4507          4620  6                  !
```

```
;4508    4621  6                [TOKEN$K_RADIX_BIN]:
;4509    4622  6                    EXPRESSION_RADIX = DBG$K_BINARY;
;4510    4623  6
;4511    4624  6
;4512    4625  6                ! Infix NOT should not show up here.
;4513    4626  6                !
;4514    4627  6                [TOKEN$K_INFIX_NOT]:
;4515    4628  6                    SIGNAL(DBG$_MISOPEMIS, 1, LEFT_OP[TOKEN$B_OPLEN]);
;4516    4629  6
;4517    4630  6                ! Any other case is an error--a lexical operator should
;4518    4631  6                ! not be on the stack or should not be marked as lexical
;4519    4632  6                ! if it has no semantic action here.
;4520    4633  6                !
;4521    4634  6                [OTHERWISE]:
;4522    4635  6                    $DBG_ERROR('DBGPARSER\EXPRESSION_PARSER 20');
;4523    4636  6
;4524    4637  6                TES;
;4525    4638  6
;4526    4639  6            END
;4527    4640  6
;4528    4641  6
;4529    4642  6        ! This is not a lexical operator--it is a "normal" operator.
;4530    4643  6        ! If this is a DEBUG Address Expression, evaluate the Address
;4531    4644  6        ! Expression operator.
;4532    4645  6        !
;4533    4646  5        ELSE IF .ADDRESS_EXPRESSION
;4534    4647  5        THEN
;4535    4648  5            OPERAND_STACK[.OPAND_INDEX] =
;4536    4649  5                DBG$EVAL_ADDR_OPERATOR(.LEFT_OP, .LEFT_ARG, .RIGHT_ARG)
;4537    4650  5
;4538    4651  5
;4539    4652  5        ! And if it is a language expression, evaluate this non-lexical
;4540    4653  5        ! operator according to language rules.
;4541    4654  5        !
;4542    4655  5        ELSE
;4543    4656  6            BEGIN
;4544    4657  6
;4545    4658  6            ! EVAL_LANG_OPERATOR returns a pointer to a Primary Descriptor
;4546    4659  6            ! or a pointer to a Value Descriptor.
;4547    4660  6            !
;4548    4661  6            PRIMPTR = DBG$EVAL_LANG_OPERATOR(.LEFT_OP, .LEFT_ARG, .RIGHT_ARG);
;4549    4662  6
;4550    4663  6
;4551    4664  6            ! If a Primary Descriptor was returned from the expression
;4552    4665  6            ! evaluator then there may be some further processing to do.
;4553    4666  6            ! An example of this is the C expression:
;4554    4667  6            ! EVALUATE  (*PTR).COMPONENT
;4555    4668  6            ! The (*PTR) will be evaluated by the expression evaluator
;4556    4669  6            ! and a Primary Descriptor will be returned (presumably,
;4557    4670  6            ! describing a record). Then we need to call the Primary
;4558    4671  6            ! Parser to pick up the rest of the expression.
;4559    4672  6            !
;4560    4673  6            IF .PRIMPTR[DBG$B_DHDR_TYPE] EQL DBG$K_PRIMARY_DESC
;4561    4674  6            THEN
;4562    4675  7                BEGIN
;4563    4676  7                DBG$PRIMARY_PARSER (
;4564    4677  7                    FALSE,              ! Operand not expected
```

```
;4565     4678  7                    FALSE,                  ! Not address expression
;4566     4679  7                    .TERM_LIST,             ! Pass along terminator list
;4567     4680  7                    0,                      ! Parenthesis nesting
;4568     4681  7                    NEW_PRIMPTR,            ! Address to fill in result
;4569     4682  7                    JUNK,                   ! Not used
;4570     4683  7                    .PRIMPTR,               ! Input Primary
;4571     4684  7                    REMEMBER_C_STATE_GOT_SUBSCRIPT);
;4572     4685  7                PRIMPTR = .NEW_PRIMPTR;
;4573     4686  6                END;
;4574     4687  6
;4575     4688  6
;4576     4689  6                ! Now just put the result of the expression on top of
;4577     4690  6                ! the expression stack.
;4578     4691  6                !
;4579     4692  6                OPERAND_STACK[.OPAND_INDEX] = .PRIMPTR;
;4580     4693  5                END;
;4581     4694  4            END;                        ! End of pop and evaluate loop
;4582     4695  4
;4583     4696  4
;4584     4697  4        ! If the new operator is a "lexical operator", meaning that it has
;4585     4698  4        ! some semantic effect on the lexical processing or parsing of the
;4586     4699  4        ! current expression, we perform that semantic action here.  For
;4587     4700  4        ! example, the terminator operator has the semantic effect of term-
;4588     4701  4        ! inating the scan of the current expression.
;4589     4702  4        !
;4590     4703  4        IF .TOKEN[TOKEN$V_LEXICAL]
;4591     4704  4        THEN
;4592     4705  5            BEGIN
;4593     4706  5            SELECTONE .TOKEN[TOKEN$W_CODE] OF
;4594     4707  5                SET
;4595     4708  5
;4596     4709  5
;4597     4710  5                ! If this is the terminator operator, exit the parse loop.
;4598     4711  5                !
;4599     4712  5                [TOKEN$K_TERMINATOR]:
;4600     4713  5                    EXITLOOP;
;4601     4714  5
;4602     4715  5
;4603     4716  5                ! If this is an open parenthesis "(", increment the
;4604     4717  5                ! parenthesis count.
;4605     4718  5                !
;4606     4719  5                [TOKEN$K_OPENPAREN]:
;4607     4720  5                    PAREN_NESTING = .PAREN_NESTING + 1;
;4608     4721  5
;4609     4722  5
;4610     4723  5                ! If this is a close parenthesis ")", decrement the paren-
;4611     4724  5                ! thesis count, check for balanced parentheses, and remove
;4612     4725  5                ! both the close parenthesis and the matching open paren-
;4613     4726  5                ! thesis from the operator stack.
;4614     4727  5                !
;4615     4728  5                [TOKEN$K_CLOSEPAREN]:
;4616     4729  6                    BEGIN
;4617     4730  6                    PAREN_NESTING = .PAREN_NESTING - 1;
;4618     4731  6                    IF .PAREN_NESTING LSS 0 THEN SIGNAL(DBG$_UNBPAREN);
;4619     4732  6                    IF .OPTOR_INDEX LSS 1
;4620     4733  6                    THEN
;4621     4734  6                        $DBG_ERROR('DBGPARSER\EXPRESSION_PARSER 2');
```

```
;  4622          4735  6
;  4623          4736  6                OPTOR_INDEX = .OPTOR_INDEX - 2;
;  4624          4737  6                TOKEN = .OPERATOR_STACK[.OPTOR_INDEX + 1];
;  4625          4738  5                END;
;  4626          4739  5
;  4627          4740  5
;  4628          4741  5            ! If this is a set constant [1, 3..4, 10], call routine
;  4629          4742  5            ! to build a value descriptor and push it on the operand
;  4630          4743  5            ! stack.  Ignore '[' by not pushing it on the operator
;  4631          4744  5            ! stack.
;  4632          4745  5            !
;  4633          4746  5            [TOKEN$K_OPENSET]:
;  4634          4747  6                BEGIN
;  4635          4748  6                OPERAND_EXPECTED = FALSE;
;  4636          4749  6                OPAND_INDEX = .OPAND_INDEX + 1;
;  4637          4750  6                IF .OPAND_INDEX GEQ MAX_OPAND_INDEX
;  4638          4751  6                THEN
;  4639          4752  6                    SIGNAL(DBG$_PARSTKOVR);
;  4640          4753  6
;  4641          4754  6                OPERAND_STACK[.OPAND_INDEX] = GET_SET_CONSTANT();
;  4642          4755  6                IF .DBG$GL_DEVELOPER[3]
;  4643          4756  6                THEN
;  4644          4757  6                    DUMP_PRIMARY(.OPERAND_STACK[.OPAND_INDEX]);
;  4645          4758  6
;  4646          4759  6                TOKEN = .OPERATOR_STACK[.OPTOR_INDEX];
;  4647          4760  6                OPTOR_INDEX = .OPTOR_INDEX - 1;
;  4648          4761  5                END;
;  4649          4762  5
;  4650          4763  5
;  4651          4764  5            ! If this is the bit-selection operator <pos,size,ext>
;  4652          4765  5            ! allowed in Address Expressions and BLISS expressions,
;  4653          4766  5            ! parse the operator's parameter fields (i.e., pick up
;  4654          4767  5            ! pos, size, and ext) and incorporate those values in the
;  4655          4768  5            ! Operator Lexical Token Entry put on the operator stack.
;  4656          4769  5            !
;  4657          4770  5            [TOKEN$K_BITSELECT]:
;  4658          4771  6                BEGIN
;  4659          4772  6                TEMP_TOKEN = DBG$GET_TEMPMEM (
;  4660          4773  6                    TOKEN$K_FIXED_SIZE_LONG +
;  4661          4774  6                    .TOKEN[TOKEN$B_OPLEN]/4);
;  4662          4775  6                CH$MOVE (
;  4663          4776  6                    TOKEN$K_FIXED_SIZE_BYTE + .TOKEN[TOKEN$B_OPLEN],
;  4664          4777  6                    .TOKEN, .TEMP_TOKEN);
;  4665          4778  6                TOKEN = .TEMP_TOKEN;
;  4666          4779  6                GET_FIELDREF(.TOKEN);
;  4667          4780  6                TOKEN[TOKEN$V_LEXICAL] = FALSE;
;  4668          4781  5                END;
;  4669          4782  5
;  4670          4783  5
;  4671          4784  5            ! Set the current radix to decimal.  Here we put a lexical
;  4672          4785  5            ! operator on the operator stack which will restore the
;  4673          4786  5            ! current radix.  We then set the radix to decimal.
;  4674          4787  5            !
;  4675          4788  5            [TOKEN$K_RADIX_DEC]:
;  4676          4789  6                BEGIN
;  4677          4790  6                TOKEN = OPERATOR_TO_RESTORE_RADIX();
;  4678          4791  6                EXPRESSION_RADIX = DBG$K_DECIMAL;
```

```
;  4679          4792  5                END;
;  4680          4793  5
;  4681          4794  5
;  4682          4795  5            ! Set the current radix to hexadecimal.  Here we put a lex-
;  4683          4796  5            ! ical operator on the operator stack which will restore
;  4684          4797  5            ! the current radix.  We then set the radix to hexadecimal.
;  4685          4798  5            !
;  4686          4799  5            [TOKEN$K_RADIX_HEX]:
;  4687          4800  6                BEGIN
;  4688          4801  6                TOKEN = OPERATOR_TO_RESTORE_RADIX();
;  4689          4802  6                EXPRESSION_RADIX = DBG$K_HEX;
;  4690          4803  5                END;
;  4691          4804  5
;  4692          4805  5
;  4693          4806  5            ! Set the current radix to octal.  Here we put a lexical
;  4694          4807  5            ! operator on the operator stack which will restore the
;  4695          4808  5            ! current radix.  We then set the radix to octal.
;  4696          4809  5            !
;  4697          4810  5            [TOKEN$K_RADIX_OCT]:
;  4698          4811  6                BEGIN
;  4699          4812  6                TOKEN = OPERATOR_TO_RESTORE_RADIX();
;  4700          4813  6                EXPRESSION_RADIX = DBG$K_OCTAL;
;  4701          4814  5                END;
;  4702          4815  5
;  4703          4816  5
;  4704          4817  5            ! Set the current radix to binary.  Here we put a lexical
;  4705          4818  5            ! operator on the operator stack which will restore the
;  4706          4819  5            ! current radix.  We then set the radix to binary.
;  4707          4820  5            !
;  4708          4821  5            [TOKEN$K_RADIX_BIN]:
;  4709          4822  6                BEGIN
;  4710          4823  6                TOKEN = OPERATOR_TO_RESTORE_RADIX();
;  4711          4824  6                EXPRESSION_RADIX = DBG$K_BINARY;
;  4712          4825  5                END;
;  4713          4826  5
;  4714          4827  5
;  4715          4828  5            ! Check for double-token (NOT =, NOT <, or NOT > in
;  4716          4829  5            ! COBOL, where NOT is marked as infix and =, >, <
;  4717          4830  5            ! is marked as prefix and lexical bit is set.  Now
;  4718          4831  5            ! at this point, we should have infix NOT on the
;  4719          4832  5            ! operator stack, and have prefix token on hand.
;  4720          4833  5            ! We'll replace the infix NOT on the operator stack
;  4721          4834  5            ! to infix "NOT <".
;  4722          4835  5            !
;  4723          4836  5            [TOKEN$K_PREFIX_EQL]:
;  4724          4837  6                BEGIN
;  4725          4838  6                TEMP_TOKEN = .OPERATOR_STACK[.OPTOR_INDEX];
;  4726          4839  6                IF .TEMP_TOKEN[TOKEN$W_CODE] EQL TOKEN$K_INFIX_NOT
;  4727          4840  6                THEN
;  4728          4841  7                    BEGIN
;  4729          4842  7                    OPTOR_INDEX = .OPTOR_INDEX - 1;
;  4730          4843  7                    TOKEN = COBOL_NOT_EQL_TOKEN;
;  4731          4844  7                    END
;  4732          4845  7
;  4733          4846  6                ELSE
;  4734          4847  6                    SIGNAL(DBG$_MISOPEMIS, 1, TOKEN[TOKEN$B_OPLEN]);
;  4735          4848  5                END;
```

```
;  4736          4849  5
;  4737          4850  5
;  4738          4851  5                    ! NOT > (Less equal)
;  4739          4852  5                    !
;  4740          4853  5                    [TOKEN$K_PREFIX_GTR]:
;  4741          4854  6                        BEGIN
;  4742          4855  6                        TEMP_TOKEN = .OPERATOR_STACK[.OPTOR_INDEX];
;  4743          4856  6                        IF .TEMP_TOKEN[TOKEN$W_CODE] EQL TOKEN$K_INFIX_NOT
;  4744          4857  6                        THEN
;  4745          4858  7                            BEGIN
;  4746          4859  7                            OPTOR_INDEX = .OPTOR_INDEX - 1;
;  4747          4860  7                            TOKEN = COBOL_NOT_GTR_TOKEN;
;  4748          4861  7                            END
;  4749          4862  7
;  4750          4863  6                        ELSE
;  4751          4864  6                            SIGNAL(DBG$_MISOPEMIS, 1, TOKEN[TOKEN$B_OPLEN]);
;  4752          4865  5                        END;
;  4753          4866  5
;  4754          4867  5
;  4755          4868  5                    ! NOT < (Greater equal)
;  4756          4869  5                    !
;  4757          4870  5                    [TOKEN$K_PREFIX_LSS]:
;  4758          4871  6                        BEGIN
;  4759          4872  6                        TEMP_TOKEN = .OPERATOR_STACK[.OPTOR_INDEX];
;  4760          4873  6                        IF .TEMP_TOKEN[TOKEN$W_CODE] EQL TOKEN$K_INFIX_NOT
;  4761          4874  6                        THEN
;  4762          4875  7                            BEGIN
;  4763          4876  7                            OPTOR_INDEX = .OPTOR_INDEX - 1;
;  4764          4877  7                            TOKEN = COBOL_NOT_LSS_TOKEN;
;  4765          4878  7                            END
;  4766          4879  7
;  4767          4880  6                        ELSE
;  4768          4881  6                            SIGNAL(DBG$_MISOPEMIS, 1, TOKEN[TOKEN$B_OPLEN]);
;  4769          4882  6
;  4770          4883  5                        END;
;  4771          4884  5
;  4772          4885  5
;  4773          4886  5                    ! Do nothing, stack.
;  4774          4887  5                    !
;  4775          4888  5                    [TOKEN$K_INFIX_NOT]:
;  4776          4889  6                        BEGIN
;  4777          4890  6                        0;
;  4778          4891  5                        END;
;  4779          4892  5
;  4780          4893  5
;  4781          4894  5                    ! Any other case is an error because the operator should
;  4782          4895  5                    ! not be marked as a "lexical" operator if it has no
;  4783          4896  5                    ! semantic action at this point.
;  4784          4897  5                    !
;  4785          4898  5                    [OTHERWISE]:
;  4786          4899  5                        $DBG_ERROR('DBGPARSER\EXPRESSION_PARSER 3');
;  4787          4900  5
;  4788          4901  5                    TES;
;  4789          4902  5
;  4790          4903  4                END;                    ! End of Lexical Operator processing
;  4791          4904  4
;  4792          4905  4
```

```
;  4793          4906  4              ! Stack the current operator on the operator stack.  Then loop to
;  4794          4907  4              ! get the next operator or operand.
;  4795          4908  4              !
;  4796          4909  4              OPTOR_INDEX = .OPTOR_INDEX + 1;
;  4797          4910  4              IF .OPTOR_INDEX GEQ MAX_OPTOR_INDEX THEN SIGNAL(DBG$_PARSTKOVR);
;  4798          4911  4              OPERATOR_STACK[.OPTOR_INDEX] = .TOKEN;
;  4799          4912  4
;  4800          4913  3              END;                            ! End of ELSE-clause for operators
;  4801          4914  3
;  4802          4915  2          END;                                ! End of the get-symbol loop
;  4803          4916  2
;  4804          4917  2
;  4805          4918  2          ! We are all done parsing the expression. Retrieve the descriptor from
;  4806          4919  2          ! the top of the stack.
;  4807          4920  2          !
;  4808          4921  2          IF .OPAND_INDEX GTR 0 THEN $DBG_ERROR('DBGPARSER\EXPRESSION_PARSER 3');
;  4809          4922  2          VALPTR = .OPERAND_STACK[0];
;  4810          4923  2
;  4811          4924  2
;  4812          4925  2          ! If this is a language expression, then we always return a Value Descriptor.
;  4813          4926  2          ! Primary Descriptors or Volati'e Value Descriptors are converted to
;  4814          4927  2          ! Value Descriptors here. DBG$EVAL_LANG_OPERATOR does this for us.
;  4815          4928  2          !
;  4816          4929  2          IF NOT .ADDRESS_EXPRESSION AND
;  4817          4930  2             NOT .DEPOSIT_FLAG
;  4818          4931  2          THEN
;  4819          4932  2              VALPTR = DBG$EVAL_LANG_OPERATOR (DBG$GL_IDENTITY_TOKEN, .VALPTR, 0);
;  4820          4933  2
;  4821          4934  2
;  4822          4935  2          ! If this is an address expression then we always return either a
;  4823          4936  2          ! Primary Descriptor or a Volatile Value Descriptor.
;  4824          4937  2          ! DBG$EVAL_ADDR_OPERATOR does this for us.
;  4825          4938  2          !
;  4826          4939  2          IF .ADDRESS_EXPRESSION
;  4827          4940  2          THEN
;  4828          4941  2              VALPTR = DBG$EVAL_ADDR_OPERATOR (DBG$GL_IDENTITY_TOKEN, .VALPTR, 0);
;  4829          4942  2
;  4830          4943  2
;  4831          4944  2          RETURN .VALPTR;
;  4832          4945  1          END;


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

24 47 42 44 5C 45 53 52 41 50 4E 47 42 44 1F 03056 P.AWS:  .ASCII  <31>\DBGNPARSE\<92>\DBG$NPARSE_EXPRESS\
   53 53 45 52 50 58 45 5F 45 53 52 41 50 4E 03065
                                             4E 4F 49 03073          .ASCII  \ION\
                        65 72 65 68 77 65 6D 6F 73 09 03076 P.AWT:  .ASCII  <9>\somewhere\
52 50 58 45 5C 52 45 53 52 41 50 47 42 44 1D 03080 P.AWU:  .ASCII  <29>\DBGPARSER\<92>\EXPRESSION_PARSER \
   20 52 45 53 52 41 50 5F 4E 4F 49 53 53 45 0308F
                                             31 0309D          .ASCII  \1\
52 50 58 45 5C 52 45 53 52 41 50 47 42 44 1E 0309E P.AWV:  .ASCII  <30>\DBGPARSER\<92>\EXPRESSION_PARSER \
   20 52 45 53 52 41 50 5F 4E 4F 49 53 53 45 030AD
                                             30 32 030BB          .ASCII  \20\
52 50 58 45 5C 52 45 53 52 41 50 47 42 44 1D 030BD P.AWW:  .ASCII  <29>\DBGPARSER\<92>\EXPRESSION_PARSER \
   20 52 45 53 52 41 50 5F 4E 4F 49 53 53 45 030CC
```

```
                                                     32  030DA                .ASCII  \2\
52 50 58 45 5C 52 45 53 52 41 50 47 42 44            1D  030DB P.AWX:     .ASCII  <29>\DBGPARSER\<92>\EXPRESSION_PARSER \
   20 52 45 53 52 41 50 5F 4E 4F 49 53 53 45             030EA
                                                     33  030F8                .ASCII  \3\
52 50 58 45 5C 52 45 53 52 41 50 47 42 44            1D  030F9 P.AWY:     .ASCII  <29>\DBGPARSER\<92>\EXPRESSION_PARSER \
   20 52 45 53 52 41 50 5F 4E 4F 49 53 53 45             03108
                                                     33  03116                .ASCII  \3\


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

                                                 0FFC 00000               .ENTRY  DBG$EXPRESSION_PARSER, Save R2,R3,R4,R5,R6,-;  4336
                                                                                  R7,R8,R9,R10,R11
                                5E      FF18     CE  9E 00002             MOVAB   -232(SP), SP
                                                 7E  D4 00007             CLRL    DEPOSIT_FLAG                                  ;  4449
                                02               6C  91 00009             CMPB    (AP), #2                                      ;  4450
                                                 1E  1B 0000C             BLEQU   2$
                                05       0C      AC  D1 0000E             CMPL    12(AP), #5                                    ;  4453
                                                 15  13 00012             BEQL    1$
                                   00000000'     EF  9F 00014             PUSHAB  P.AWS                                         ;  4455
                                                 01  DD 0001A             PUSHL   #1
                                   00028362      8F  DD 0001C             PUSHL   #164706
                          00000000G  00          03  FB 00022             CALLS   #3, LIB$SIGNAL
                                6E               01  D0 00029 1$:         MOVL    #1, DEPOSIT_FLAG                              ;  4457
                                                 56  D4 0002C 2$:         CLRL    OPTOR_INDEX                                   ;  4465
                             24 AE46 00000000'   EF  9E 0002E             MOVAB   INITIATOR_TOKEN, OPERATOR_STACK-              ;  4466
                                                                                  [OPTOR_INDEX]
                                57               01  CE 00037             MNEGL   #1, OPAND_INDEX                               ;  4467
                             04 AE               01  D0 0003A             MOVL    #1, OPERAND_EXPECTED                          ;  4468
                                                 5B  D4 0003E             CLRL    PAREN_NESTING                                 ;  4469
                                      14 AE      9F  00040 3$:            PUSHAB  TOKEN_OPERAND_FLAG                            ;  4485
                                      1C AE      9F  00043                PUSHAB  TOKEN
                                                 5B  DD 00046             PUSHL   PAREN_NESTING                                 ;  4486
                                7E       04 AC     7D  00048             MOVQ    ADDRESS_EXPRESSION, -(SP)                     ;  4485
                                      18 AE      DD  0004C                PUSHL   OPERAND_EXPECTED
                          0000V CF               06  FB 0004F             CALLS   #6, DBG$PRIMARY_PARSER
                                58       18 AE     D0  00054             MOVL    TOKEN, R8                                     ;  4504
                                46       14 AE     E9  00058             BLBC    TOKEN_OPERAND_FLAG, 6$                        ;  4493
                                15       04 AE     E8  0005C             BLBS    OPERAND_EXPECTED, 4$                          ;  4496
                                   00000000'     EF  9F 00060             PUSHAB  P.AWT                                         ;  4499
                                                 01  DD 00066             PUSHL   #1
                                   000289AA      8F  DD 00068             PUSHL   #166314
                          00000000G  00          03  FB 0006E             CALLS   #3, LIB$SIGNAL
                                      04 AE      D4  00075 4$:            CLRL    OPERAND_EXPECTED                              ;  4501
                                                 57  D6 00078             INCL    OPAND_INDEX                                   ;  4502
                                19               57  D1 0007A             CMPL    OPAND_INDEX, #25                              ;  4503
                                                 0D  19 0007D             BLSS    5$
                                   000280E0      8F  DD 0007F             PUSHL   #164064
                          00000000G  00          01  FB 00085             CALLS   #1, LIB$SIGNAL
                                9C AD47          58  D0 0008C 5$:         MOVL    R8, OPERAND_STACK[OPAND_INDEX]                ;  4504
                       A7 00000000G  00          03  E1 00091             BBC     #3, DBG$GL_DEVELOPER, 3$                      ;  4505
                                                 58  DD 00099             PUSHL   R8
                          0000V CF               01  FB 0009B             CALLS   #1, DUMP_PRIMARY
                                                 9E  11 000A0             BRB     3$                                            ;  4493
                                0A       04 AE     E9  000A2 6$:            BLBC    OPERAND_EXPECTED, 7$                          ;  4525
                                02       18 BE     91  000A6                CMPB    @TOKEN, #2                                    ;  4526
```

```
                        0A  12 000AA          BNEQ    8$                                                      ;
                  1A    04  AE E8 000AC          BLBS    OPERAND_EXPECTED, 9$                                    ; 4527
                  02    18  BE 91 000B0 7$:     CMPB    @TOKEN, #2                                              ; 4528
                        14  12 000B4          BNEQ    9$                                                      ;
7E          18    AE        0C C1 000B6 8$:     ADDL3   #12, TOKEN, -(SP)                                       ; 4530
                        01  DD 000BB          PUSHL   #1                                                      ;
                000289B2    8F DD 000BD          PUSHL   #166322                                                 ;
    00000000G   00          03 FB 000C3          CALLS   #3, LIB$SIGNAL                                          ;
                04          68 91 000CA 9$:     CMPB    (R8), #4                                                ; 4532
                            04 13 000CD          BEQL    10$                                                     ;
          04    AE          01 D0 000CF          MOVL    #1, OPERAND_EXPECTED                                    ; 4534
                59    24 AE46 D0 000D3 10$:     MOVL    OPERATOR_STACK[OPTOR_INDEX], LEFT_OP                    ; 4549
          05    A8    04    A9 91 000D8          CMPB    4(LEFT_OP), 5(R8)                                       ; 4550
                            03 1E 000DD          BGEQU   11$                                                     ;
                        0121 31 000DF          BRW     28$                                                     ;
                15          56 F4 000E2 11$:     SOBGEQ  OPTOR_INDEX, 12$                                        ; 4559
                00000000'   EF 9F 000E5          PUSHAB  P.AWU                                                   ; 4562
                            01 DD 000EB          PUSHL   #1                                                      ;
                00028362    8F DD 000ED          PUSHL   #164706                                                 ;
    00000000G   00          03 FB 000F3          CALLS   #3, LIB$SIGNAL                                          ;
          08    AE    9C AD47 D0 000FA 12$:     MOVL    OPERAND_STACK[OPAND_INDEX], LEFT_ARG                    ; 4564
                0C    AE    D4 00100          CLRL    RIGHT_ARG                                               ; 4565
                03          69 91 00103          CMPB    (LEFT_OP), #3                                           ; 4566
                            0D 12 00106          BNEQ    13$                                                     ;
          0C    AE    08    AE D0 00108          MOVL    LEFT_ARG, RIGHT_ARG                                     ; 4569
                            57 D7 0010D          DECL    OPAND_INDEX                                             ; 4570
          08    AE    9C AD47 D0 0010F          MOVL    OPERAND_STACK[OPAND_INDEX], LEFT_ARG                    ; 4571
09 00000000G    00          03 E1 00115 13$:     BBC     #3, DBG$GL_DEVELOPER, 14$                               ; 4574
                            7E D4 0011D          CLRL    -(SP)                                                   ;
                            59 DD 0011F          PUSHL   LEFT_OP                                                 ;
        0000V   CF          02 FB 00121          CALLS   #2, DUMP_OPERATOR                                       ;
7B              69          09 E1 00126 14$:     BBC     #9, (LEFT_OP), 24$                                      ; 4582
                0B    02    A9 B1 0012A          CMPW    2(LEFT_OP), #11                                         ; 4593
                            0F 12 0012E          BNEQ    16$                                                     ;
                000289FA    8F DD 00130          PUSHL   #166394                                                 ; 4594
    00000000G   00          01 FB 00136          CALLS   #1, LIB$SIGNAL                                          ;
                            94 11 0013D 15$:     BRB     10$                                                     ;
                34    02    A9 B1 0013F 16$:     CMPW    2(LEFT_OP), #52                                         ; 4600
                            09 12 00143          BNEQ    18$                                                     ;
    00000000'   EF          0A D0 00145          MOVL    #10, EXPRESSION_RADIX                                   ; 4601
                            85 11 0014C 17$:     BRB     10$                                                     ;
                35    02    A9 B1 0014E 18$:     CMPW    2(LEFT_OP), #53                                         ; 4607
                            09 12 00152          BNEQ    19$                                                     ;
    00000000'   EF          10 D0 00154          MOVL    #16, EXPRESSION_RADIX                                   ; 4608
                            60 11 0015B          BRB     25$                                                     ;
                36    02    A9 B1 0015D 19$:     CMPW    2(LEFT_OP), #54                                         ; 4614
                            09 12 00161          BNEQ    20$                                                     ;
    00000000'   EF          08 D0 00163          MOVL    #8, EXPRESSION_RADIX                                    ; 4615
                            51 11 0016A          BRB     25$                                                     ;
                37    02    A9 B1 0016C 20$:     CMPW    2(LEFT_OP), #55                                         ; 4621
                            09 12 00170          BNEQ    21$                                                     ;
    00000000'   EF          02 D0 00172          MOVL    #2, EXPRESSION_RADIX                                    ; 4622
                            42 11 00179          BRB     25$                                                     ;
                2C    02    A9 B1 0017B 21$:     CMPW    2(LEFT_OP), #44                                         ; 4627
                            0D 12 0017F          BNEQ    22$                                                     ;
                      0C    A9 9F 00181          PUSHAB  12(LEFT_OP)                                             ; 4628
                            01 DD 00184          PUSHL   #1                                                      ;
```

```
                                    000289B2   8F   DD 00186          PUSHL   #166322                                    ;
                                               0E   11 0018C          BRB     23$                                        ;
                                    00000000'  EF   9F 0018E 22$:     PUSHAB  P.AWV                                      ; 4635
                                               01   DD 00194          PUSHL   #1                                         ;
                                    00028362   8F   DD 00196          PUSHL   #164706                                    ;
                        00000000G   00              03   FB 0019C 23$:     CALLS   #3, LIB$SIGNAL                        ;
                                               98   11 001A3          BRB     15$                                        ; 4582
                                 16        04   AC   E9 001A5 24$:     BLBC    ADDRESS_EXPRESSION, 26$                    ; 4649
                                           0C   AE   DD 001A9          PUSHL   RIGHT_ARG                                  ;
                                           0C   AE   DD 001AC          PUSHL   LEFT_ARG                                   ;
                                               59   DD 001AF          PUSHL   LEFT_OP                                    ;
                        00000000G   00              03   FB 001B1          CALLS   #3, DBG$EVAL_ADDR_OPERATOR            ;
                              9C AD47               50   D0 001B8          MOVL    R0, OPERAND_STACK[OPAND_INDEX]        ;
                                               8D   11 001BD 25$:     BRB     17$                                        ; 4668
                                           0C   AE   DD 001BF 26$:     PUSHL   RIGHT_ARG                                  ; 4661
                                           0C   AE   DD 001C2          PUSHL   LEFT_ARG                                   ;
                                               59   DD 001C5          PUSHL   LEFT_OP                                    ;
                        00000000G   00              03   FB 001C7          CALLS   #3, DBG$EVAL_LANG_OPERATOR            ;
                                 10   AE            50   D0 001CE          MOVL    R0, PRIMPTR                           ;
00000079   8F         10   BE        08            10   ED 001D2          CMPZV   #16, #8, @PRIMPTR, #121               ; 4673
                                               1C   12 001DC          BNEQ    27$                                        ;
                                               16   DD 001DE          PUSHL   #22                                        ; 4676
                                           14   AE   DD 001E0          PUSHL   PRIMPTR                                    ; 4683
                                           24   AE   9F 001E3          PUSHAB  JUNK                                       ; 4676
                                           2C   AE   9F 001E6          PUSHAB  NEW_PRIMPTR                                ;
                                               7E   D4 001E9          CLRL    -(SP)                                      ;
                                           08   AC   DD 001EB          PUSHL   TERM_LIST                                  ; 4679
                                               7E   7C 001EE          CLRQ    -(SP)                                      ; 4676
                              0000V   CF            08   FB 001F0          CALLS   #8, DBG$PRIMARY_PARSER                ;
                                 10   AE        20   AE   D0 001F5          MOVL    NEW_PRIMPTR, PRIMPTR                  ; 4685
                                 9C AD47       10   AE   D0 001FA 27$:     MOVL    PRIMPTR, OPERAND_STACK[OPAND_INDEX]   ; 4692
                                          FED0      31   00200          BRW     10$                                        ; 4541
                     4B                    68        09   E1 00203 28$:     BBC     #9, (R8), 33$                          ; 4703
                                 02        02   A8   B1 00207          CMPW    2(R8), #2                                  ; 4712
                                               03   12 0020B          BNEQ    29$                                        ;
                                          01BF      31   0020D          BRW     56$                                        ;
                                 0B        02   A8   B1 00210 29$:     CMPW    2(R8), #11                                 ; 4719
                                               04   12 00214          BNEQ    30$                                        ;
                                               5B   D6 00216          INCL    PAREN_NESTING                              ; 4720
                                               7A   11 00218          BRB     37$                                        ;
                                 0C        02   A8   B1 0021A 30$:     CMPW    2(R8), #12                                 ; 4728
                                               34   12 0021E          BNEQ    34$                                        ;
                                 0D            5B   F4 00220          SOBGEQ  PAREN_NESTING, 31$                         ; 4730
                                    000289FA   8F   DD 00223          PUSHL   #166394                                    ; 4731
                        00000000G   00              01   FB 00229          CALLS   #1, LIB$SIGNAL                        ;
                                               56   D5 00230 31$:     TSTL    OPTOR_INDEX                                ; 4732
                                               15   14 00232          BGTR    32$                                        ;
                                    00000000'  EF   9F 00234          PUSHAB  P.AWW                                      ; 4734
                                               01   DD 0023A          PUSHL   #1                                         ;
                                    00028362   8F   DD 0023C          PUSHL   #164706                                    ;
                        00000000G   00              03   FB 00242          CALLS   #3, LIB$SIGNAL                        ;
                                 56            02   C2 00249 32$:     SUBL2   #2, OPTOR_INDEX                            ; 4736
                              18   AE        28 AE46   D0 0024C          MOVL    OPERATOR_STACK+4[OPTOR_INDEX], TOKEN       ; 4737
                                               79   11 00252 33$:     BRB     39$                                        ; 4706
                                 39        02   A8   B1 00254 34$:     CMPW    2(R8), #57                                 ; 4746
                                               3C   12 00258          BNEQ    38$                                        ;
                                           04   AE   D4 0025A          CLRL    OPERAND_EXPECTED                           ; 4748
```

```
                                57  D6 0025D          INCL    OPAND_INDEX                                     ; 4749
                             19 57  D1 0025F          CMPL    OPAND_INDEX, #25                                ; 4750
                                0D  19 00262          BLSS    35$
                      000280E0  8F  DD 00264          PUSHL   #164064                                         ; 4752
          00000000G   00        01  FB 0026A          CALLS   #1, LIB$SIGNAL
               0000V  CF        00  FB 00271 35$:     CALLS   #0, GET_SET_CONSTANT                            ; 4754
                   9C AD47      50  D0 00276          MOVL    R0, OPERAND_STACK[OPAND_INDEX]
       09 00000000G   00        03  E1 0027B          BBC     #3, DBG$GL_DEVELOPER, 36$                       ; 4755
                      9C AD47   DD  00283             PUSHL   OPERAND_STACK[OPAND_INDEX]                      ; 4757
               0000V  CF        01  FB 00287          CALLS   #1, DUMP_PRIMARY
                 18   AE     24 AE46  D0 0028C 36$:     MOVL    OPERATOR_STACK[OPTOR_INDEX], TOKEN              ; 4759
                                56  D7 00292          DECL    OPTOR_INDEX                                     ; 4760
                                7F  11 00294 37$:     BRB     43$                                             ; 4706
                             31 02  A8  B1 00296 38$:     CMPW    2(R8), #49                                      ; 4770
                                33  12 0029A          BNEQ    40$
                             50 0C  A8  9A 0029C          MOVZBL  12(R8), R0                                      ; 4774
                             50     04  C6 002A0          DIVL2   #4, R0
                                04  A0  9F 002A3          PUSHAB  4(R0)                                           ; 4773
          00000000G   00        01  FB 002A6          CALLS   #1, DBG$GET_TEMPMEM
                             5A 50  D0 002AD          MOVL    R0, TEMP_TOKEN
                             50 0C  A8  9A 002B0          MOVZBL  12(R8), R0                                      ; 4776
                             50     0D  C0 002B4          ADDL2   #13, R0
                6A           68     50  28 002B7          MOVC3   R0, (R8), (TEMP_TOKEN)                          ; 4777
                   18   AE        5A  D0 002BB          MOVL    TEMP_TOKEN, TOKEN                               ; 4778
                                18 AE  DD 002BF          PUSHL   TOKEN                                           ; 4779
               0000V  CF        01  FB 002C2          CALLS   #1, GET_FIELDREF
                 18   BE   0200  8F  AA 002C7          BICW2   #512, @TOKEN                                    ; 4780
                                7D  11 002CD 39$:     BRB     46$                                             ; 4706
                             34 02  A8  B1 002CF 40$:     CMPW    2(R8), #52                                      ; 4788
                                12  12 002D3          BNEQ    41$
               0000V  CF        00  FB 002D5          CALLS   #0, OPERATOR_TO_RESTORE_RADIX                   ; 4790
                 18   AE        50  D0 002DA          MOVL    R0, TOKEN
          00000000'   EF        0A  D0 002DE          MOVL    #10, EXPRESSION_RADIX                           ; 4791
                                65  11 002E5          BRB     46$                                             ; 4706
                             35 02  A8  B1 002E7 41$:     CMPW    2(R8), #53                                      ; 4799
                                12  12 002EB          BNEQ    42$
               0000V  CF        00  FB 002ED          CALLS   #0, OPERATOR_TO_RESTORE_RADIX                   ; 4801
                 18   AE        50  D0 002F2          MOVL    R0, TOKEN
          00000000'   EF        10  D0 002F6          MOVL    #16, EXPRESSION_RADIX                           ; 4802
                                6A  11 002FD          BRB     48$                                             ; 4706
                             36 02  A8  B1 002FF 42$:     CMPW    2(R8), #54                                      ; 4810
                                12  12 00303          BNEQ    44$
               0000V  CF        00  FB 00305          CALLS   #0, OPERATOR_TO_RESTORE_RADIX                   ; 4812
                 18   AE        50  D0 0030A          MOVL    R0, TOKEN
          00000000'   EF        08  D0 0030E          MOVL    #8, EXPRESSION_RADIX                            ; 4813
                                71  11 00315 43$:     BRB     50$                                             ; 4706
                             37 02  A8  B1 00317 44$:     CMPW    2(R8), #55                                      ; 4821
                                12  12 0031B          BNEQ    45$
               0000V  CF        00  FB 0031D          CALLS   #0, OPERATOR_TO_RESTORE_RADIX                   ; 4823
                 18   AE        50  D0 00322          MOVL    R0, TOKEN
          00000000'   EF        02  D0 00326          MOVL    #2, EXPRESSION_RADIX                            ; 4824
                                59  11 0032D          BRB     50$                                             ; 4706
                 0041   8F  02  A8  B1 0032F 45$:     CMPW    2(R8), #65                                      ; 4836
                                17  12 00335          BNEQ    47$
                             5A 24 AE46  D0 00337          MOVL    OPERATOR_STACK[OPTOR_INDEX], TEMP_TOKEN         ; 4838
                             2C 02  AA  B1 0033C          CMPW    2(TEMP_TOKEN), #44                              ; 4839
                                48  12 00340          BNEQ    51$
```

```
                          56  D7 00342          DECL    OPTOR_INDEX                                          ; 4842
                18  AE 00000000'  EF  9E 00344          MOVAB   COBOL_NOT_EQL_TOKEN, TOKEN                   ; 4843
                          64  11 0034C 46$:      BRB     54$                                                  ; 4839
                      3F    02  A8  B1 0034E 47$:      CMPW    2(R8), #63                                   ; 4853
                          17  12 00352          BNEQ    49$
                      5A    24 AE46  D0 00354          MOVL    OPERATOR_STACK[OPTOR_INDEX], TEMP_TOKEN      ; 4855
                      2C    02  AA  B1 00359          CMPW    2(TEMP_TOKEN), #44                           ; 4856
                          2B  12 0035D          BNEQ    51$
                          56  D7 0035F          DECL    OPTOR_INDEX                                          ; 4859
                18  AE 00000000'  EF  9E 00361          MOVAB   COBOL_NOT_GTR_TOKEN, TOKEN                   ; 4860
                          47  11 00369 48$:      BRB     54$                                                  ; 4856
              0040  8F    02  A8  B1 0036B 49$:      CMPW    2(R8), #64                                   ; 4870
                          24  12 00371          BNEQ    52$
                      5A    24 AE46  D0 00373          MOVL    OPERATOR_STACK[OPTOR_INDEX], TEMP_TOKEN      ; 4872
                      2C    02  AA  B1 00378          CMPW    2(TEMP_TOKEN), #44                           ; 4873
                          0C  12 0037C          BNEQ    51$
                          56  D7 0037E          DECL    OPTOR_INDEX                                          ; 4876
                18  AE 00000000'  EF  9E 00380          MOVAB   COBOL_NOT_LSS_TOKEN, TOKEN                   ; 4877
                          28  11 00388 50$:      BRB     54$                                                  ; 4873
                          0C  A8  9F 0038A 51$:      PUSHAB  12(R8)                                       ; 4881
                          01  DD 0038D          PUSHL   #1
                   000289B2   8F  DD 0038F          PUSHL   #166322
                          14  11 00395          BRB     53$
                      2C    02  A8  B1 00397 52$:      CMPW    2(R8), #44                                   ; 4888
                          15  13 0039B          BEQL    54$
                   00000000'  EF  9F 0039D          PUSHAB  P.AWX                                        ; 4899
                          01  DD 003A3          PUSHL   #1
                   00028362   8F  DD 003A5          PUSHL   #164706
          00000000G   00          03  FB 003AB 53$:      CALLS   #3, LIB$SIGNAL
                          56  D6 003B2 54$:      INCL    OPTOR_INDEX                                          ; 4909
                      19          56  D1 003B4          CMPL    OPTOR_INDEX, #25                             ; 4910
                          0D  19 003B7          BLSS    55$
                   000280E0   8F  DD 003B9          PUSHL   #164064
          00000000G   00          01  FB 003BF          CALLS   #1, LIB$SIGNAL
                24  AE46        18  AE  D0 003C6 55$:      MOVL    TOKEN, OPERATOR_STACK[OPTOR_INDEX]           ; 4911
                          FC71  31  003CC          BRW     3$                                                   ; 4477
                          57  D5 003CF 56$:      TSTL    OPAND_INDEX                                          ; 4921
                          15  15 003D1          BLEQ    57$
                00000000'  EF  9F 003D3          PUSHAB  P.AWY
                          01  DD 003D9          PUSHL   #1
                00028362   8F  DD 003DB          PUSHL   #164706
          00000000G   00          03  FB 003E1          CALLS   #3, LIB$SIGNAL
                      50    9C  AD  D0 003E8 57$:      MOVL    OPERAND_STACK, VALPTR                        ; 4922
                      18    04  AC  E8 003EC          BLBS    ADDRESS_EXPRESSION, 59$                      ; 4929
                      11          6E  E8 003F0          BLBS    DEPOSIT_FLAG, 58$                            ; 4930
                          7E  D4 003F3          CLRL    -(SP)                                                ; 4932
                          50  DD 003F5          PUSHL   VALPTR
                00000000'  EF  9F 003F7          PUSHAB  DBG$GL_IDENTITY_TOKEN
          00000000G   00          03  FB 003FD          CALLS   #3, DBG$EVAL_LANG_OPERATOR
                      11    04  AC  E9 00404 58$:      BLBC    ADDRESS_EXPRESSION, 60$                      ; 4939
                          7E  D4 00408 59$:      CLRL    -(SP)                                                ; 4941
                          50  DD 0040A          PUSHL   VALPTR
                00000000'  EF  9F 0040C          PUSHAB  DBG$GL_IDENTITY_TOKEN
          00000000G   00          03  FB 00412          CALLS   #3, DBG$EVAL_ADDR_OPERATOR
                          04  00419 60$:      RET                                                          ; 4945

; Routine Size:  1050 bytes,    Routine Base:  DBG$CODE + 05F8
```

```
;  4834     4946  1 GLOBAL ROUTINE DBG$GET_BIF_ARGUMENTS (LENGTH, NAME) =
;  4835     4947  1 !
;  4836     4948  1 ! FUNCTION
;  4837     4949  1 !       This routine picks up built-in function arguments.  It calls
;  4838     4950  1 !       DBG$EXPRESSION_PARSER to parse and evaluate each argument
;  4839     4951  1 !       expression.
;  4840     4952  1 !
;  4841     4953  1 !       This routine assumes that the opening set parenthesis has
;  4842     4954  1 !       already been found and that the parse pointer points to the
;  4843     4955  1 !       start of the first built-in function argument expression.
;  4844     4956  1 !       When this routine returns, the parse pointer is left pointing
;  4845     4957  1 !       at the first character after the closing set parenthesis.
;  4846     4958  1 !
;  4847     4959  1 ! INPUTS
;  4848     4960  1 !       LENGTH - The number of arguments expected.
;  4849     4961  1 !       NAME   - Name of Built-in function or Ada tick operator to output
;  4850     4962  1 !                when an error is to be signaled.
;  4851     4963  1 !
;  4852     4964  1 ! OUTPUTS
;  4853     4965  1 !       Pointer to a counted vector of long words that point to the
;  4854     4966  1 !                argument values. The vector will have a minimum length
;  4855     4967  1 !                of 2 arguments.
;  4856     4968  1 !
;  4857     4969  1
;  4858     4970  2     BEGIN
;  4859     4971  2
;  4860     4972  2     MAP
;  4861     4973  2         NAME    : REF VECTOR [,BYTE];           ! Name of function or tick operator
;  4862     4974  2
;  4863     4975  2     LOCAL
;  4864     4976  2         ARG_PTR         : REF VECTOR [,LONG],   ! Pointer to counted vect. of arguments
;  4865     4977  2         BIF_INDEX;                              ! Index into the vector
;  4866     4978  2
;  4867     4979  2
;  4868     4980  2     ! Get temporary memory for the argument list.
;  4869     4981  2     !
;  4870     4982  2     ARG_PTR = DBG$GET_TEMPMEM( MAX((.LENGTH + 1), 3) );
;  4871     4983  2
;  4872     4984  2     ! Set the vector index to 1 and loop, picking up arguments, until
;  4873     4985  2     ! a closing paren is found.  The arguments are picked up by a
;  4874     4986  2     ! recursive call on DBG$EXPRESSION_PARSER.
;  4875     4987  2     !
;  4876     4988  2     BIF_INDEX = 1;
;  4877     4989  2     TERMINATOR_CODE = TOKEN$K_TERM_NONE;
;  4878     4990  2     WHILE .TERMINATOR_CODE NEQ TOKEN$K_TERM_CLOSE DO
;  4879     4991  3         BEGIN
;  4880     4992  3
;  4881     4993  3         ! If the number of arguments found so far is more than the number
;  4882     4994  3         ! expected, signal a bad argument list.
;  4883     4995  3         !
;  4884     4996  3         IF .BIF_INDEX GTR .LENGTH
;  4885     4997  3         THEN
;  4886     4998  3             SIGNAL(DBG$_INVARGLIS, 1, .NAME);
;  4887     4999  3
;  4888     5000  3         ! Recursive call to pick up the argument, and store the pointer
;  4889     5001  3         ! away in the vector.
;  4890     5002  3         !
```

```
;  4891          5003  3          ARG_PTR[.BIF_INDEX] = DBG$EXPRESSION_PARSER(FALSE, COMPAREN_TERM_TBL);
;  4892          5004  3
;  4893          5005  3          ! If an invalid terminator was found signal the error.  If not,
;  4894          5006  3          ! increment the character pointer past the terminator.
;  4895          5007  3          !
;  4896          5008  3          IF .TERMINATOR_CODE EQL TOKEN$K_TERM_NONE
;  4897          5009  3          THEN
;  4898          5010  3              SIGNAL(DBG$_MISCLOSUB);
;  4899          5011  3          CHARPTR = .CHARPTR + .TERMINATOR_LENGTH;
;  4900          5012  3          BIF_INDEX = .BIF_INDEX + 1;
;  4901          5013  3
;  4902          5014  2          END;
;  4903          5015  2
;  4904          5016  2      BIF_INDEX = .BIF_INDEX - 1;
;  4905          5017  2
;  4906          5018  2      ! If the number of arguments found is not the same as the number of
;  4907          5019  2      ! expected, signal a bad argument list.
;  4908          5020  2      !
;  4909          5021  2      IF .BIF_INDEX NEQ .LENGTH
;  4910          5022  2      THEN
;  4911          5023  2          SIGNAL(DBG$_INVARGLIS, 1, .NAME);
;  4912          5024  2
;  4913          5025  2      ! If only one argument is requested, a zero is left for the second argument.
;  4914          5026  2      ! This is done for the subsequent call to DBG$EVAL_LANG_OPERATOR.
;  4915          5027  2      ! Then store away the number of arguments and return a vector of these
;  4916          5028  2      ! arguments.
;  4917          5029  2      !
;  4918          5030  2      ARG_PTR[0] = .BIF_INDEX;
;  4919          5031  2
;  4920          5032  2      RETURN .ARG_PTR;
;  4921          5033  1      END;


                                  003C 00000          .ENTRY  DBG$GET_BIF_ARGUMENTS, Save R2,R3,R4,R5  ; 4946
                55 00000000G   00   9E 00002          MOVAB   LIB$SIGNAL, R5
                54 00000000'   EF   9E 00009          MOVAB   TERMINATOR_CODE, R4
50        04    AC             01   C1 00010          ADDL3   #1, LENGTH, R0                           ; 4982
                               50   DD 00015          PUSHL   R0
                03             6E   D1 00017          CMPL    (SP), #3
                               03   18 0001A          BGEQ    1$
                6E             03   D0 0001C          MOVL    #3, (SP)
    00000000G   00             01   FB 0001F 1$:      CALLS   #1, DBG$GET_TEMPMEM
                53             50   D0 00026          MOVL    R0, ARG_PTR
                52             01   D0 00029          MOVL    #1, BIF_INDEX                           ; 4988
                               64   D4 0002C          CLRL    TERMINATOR_CODE                         ; 4989
                02             64   D1 0002E 2$:      CMPL    TERMINATOR_CODE, #2                     ; 4990
                               3C   13 00031          BEQL    5$
          04    AC             52   D1 00033          CMPL    BIF_INDEX, LENGTH                       ; 4996
                               0E   15 00037          BLEQ    3$
                      08       AC   DD 00039          PUSHL   NAME                                    ; 4998
                               01   DD 0003C          PUSHL   #1
                00028838       8F   DD 0003E          PUSHL   #165944
                65             03   FB 00044          CALLS   #3, LIB$SIGNAL
                00000000'      EF   9F 00047 3$:      PUSHAB  COMPAREN_TERM_TBL                       ; 5003
```

```
                              7E D4 0004D        CLRL    -(SP)                                ;
          FB92  CF            02 FB 0004F        CALLS   #2, DBG$EXPRESSION_PARSER            ;
             6342             50 D0 00054        MOVL    R0, (ARG_PTR)[BIF_INDEX]             ;
                              64 D5 00058        TSTL    TERMINATOR_CODE                      ; 5008
                              09 12 0005A        BNEQ    4$                                   ;
                    00028E90  8F DD 0005C        PUSHL   #167568                              ; 5010
                65            01 FB 00062        CALLS   #1, LIB$SIGNAL                       ;
          FBCC  C4        04  A4 C0 00065 4$:    ADDL2   TERMINATOR_LENGTH, CHARPTR           ; 5011
                              52 D6 0006B        INCL    BIF_INDEX                            ; 5012
                              BF 11 0006D        BRB     2$                                   ; 4990
                              52 D7 0006F 5$:    DECL    BIF_INDEX                            ; 5016
            04  AC            52 D1 00071        CMPL    BIF_INDEX, LENGTH                    ; 5021
                              0E 13 00075        BEQL    6$                                   ;
                          08  AC DD 00077        PUSHL   NAME                                 ; 5023
                              01 DD 0007A        PUSHL   #1                                   ;
                    00028838  8F DD 0007C        PUSHL   #165944                              ;
                65            03 FB 00082        CALLS   #3, LIB$SIGNAL                       ;
                63            52 D0 00085 6$:    MOVL    BIF_INDEX, (ARG_PTR)                 ; 5030
                50            53 D0 00088        MOVL    ARG_PTR, R0                          ; 5032
                                 04 0008B        RET                                          ; 5033

; Routine Size:  140 bytes,    Routine Base:  DBG$CODE + 0A12
```

```
;  4923          5034  1 GLOBAL ROUTINE DBG$LEXICAL_SCANNER(OPERAND_EXPECTED,
;  4924          5035  1                                    ADDRESS_EXPRESSION, TERM_LIST, PAREN_NESTING) =
;  4925          5036  1 !
;  4926          5037  1 ! FUNCTION
;  4927          5038  1 !       This routine is the Lexical Scanner used during expression parsing.
;  4928          5039  1 !       It scans the character or characters pointed to by CHARPTR to pick
;  4929          5040  1 !       up the next lexical token according to the rules of the current
;  4930          5041  1 !       language.  It picks up or constructs a Lexical Token Entry for the
;  4931          5042  1 !       found lexical token and returns a pointer to that Token Entry as its
;  4932          5043  1 !       result.
;  4933          5044  1 !
;  4934          5045  1 !       The Lexical Scanner is called by the Primary Parser which then uses
;  4935          5046  1 !       the returned token to build up a Primary Descriptor if the token is
;  4936          5047  1 !       part of a Primary Symbol.  If the returned token is not part of a
;  4937          5048  1 !       Primary Symbol (i.e., if it is an operator in the current language or
;  4938          5049  1 !       the current Address Expression or if it is a constant), the Primary
;  4939          5050  1 !       Parser returns it directly to the Expression Parser.  The Expression
;  4940          5051  1 !       Parser is thus fed a stream of operands and operators which it then
;  4941          5052  1 !       uses to interpret and evaluate the current expression.
;  4942          5053  1 !
;  4943          5054  1 !       This routine assumes that the input line being scanned has already
;  4944          5055  1 !       been converted to upper case and it assumes that the input line is
;  4945          5056  1 !       terminated by a carriage-return character.  It also assumes that
;  4946          5057  1 !       the variable CHARPTR has been set up to point to the current posi-
;  4947          5058  1 !       tion in the buffer being scanned.  CHARPTR is updated by this rou-
;  4948          5059  1 !       tine to point to the first character position after the current
;  4949          5060  1 !       token.
;  4950          5061  1 !
;  4951          5062  1 !       This routine accepts a list of allowed "terminator tokens" (keywords
;  4952          5063  1 !       such as "DO" or "THEN" or special characters such as ",", ")", or "=",
;  4953          5064  1 !       depending on context).  This list is passed to the Lexical Scanner
;  4954          5065  1 !       which returns the Terminator Operator when such a token or a carriage-
;  4955          5066  1 !       return is encountered.  As a side effect, OWN variable TERMINATOR_CODE
;  4956          5067  1 !       is set to a value which indicates which terminator token was found.
;  4957          5068  1 !       That terminator's character length is also set in TERMINATOR_LENGTH.
;  4958          5069  1 !       (This side effect is used when parsing subscript expressions.)
;  4959          5070  1 !
;  4960          5071  1 ! INPUTS
;  4961          5072  1 !       OPERAND_EXPECTED - A flag set to TRUE if an operand is expected next
;  4962          5073  1 !                  in the parse of the current expression.  This flag is used
;  4963          5074  1 !                  to disambiguate certain operators, such as "+" and "-",
;  4964          5075  1 !                  which are prefix operators when an operand is expected
;  4965          5076  1 !                  next and are infix operators when an operator is expected
;  4966          5077  1 !                  next.
;  4967          5078  1 !
;  4968          5079  1 !       ADDRESS_EXPRESSION - A flag set to TRUE if we are parsing a DEBUG
;  4969          5080  1 !                  Address Expression instead of a language expression.  This
;  4970          5081  1 !                  affects the parsing of Address Expression operators such
;  4971          5082  1 !                  as "+", "-", "*", "/", ":", and "@" which are recognized by
;  4972          5083  1 !                  DEBUG rules, not language rules, in Address Expressions.
;  4973          5084  1 !
;  4974          5085  1 !       TERM_LIST - A vector of pointers to Terminator Lexical Token Entries
;  4975          5086  1 !                  for the Terminator Tokens which can terminate the expression
;  4976          5087  1 !                  to be parsed.  The vector must be in PLIT form (TERM_LIST[-1]
;  4977          5088  1 !                  gives the number of entries) and each pointer is expected to
;  4978          5089  1 !                  be relative to TABLEBASE.  If there are no terminator tokens
;  4979          5090  1 !                  other than carriage return, this list is empty (0 entries).
```

```
;  4980          5091  1 !
;  4981          5092  1 !       PAREN_NESTING - The current parenthesis nesting depth.  This parameter
;  4982          5093  1 !                       is used to detect whether certain tokens are expression
;  4983          5094  1 !                       terminators or not.  (For example, a ")" token in a sub-
;  4984          5095  1 !                       script expression terminates it only if parentheses are
;  4985          5096  1 !                       already balanced.)
;  4986          5097  1 !
;  4987          5098  1 ! OUTPUTS
;  4988          5099  1 !       A pointer to the Lexical Token Entry for the next lexical token found
;  4989          5100  1 !                       in the input line is returned as the routine value.  If there
;  4990          5101  1 !                       are no more lexical tokens on the line, a pointer to a Token
;  4991          5102  1 !                       Entry for the TOKEN$K_TERMINATOR operator is returned.
;  4992          5103  1 !
;  4993          5104  1
;  4994          5105  2     BEGIN
;  4995          5106  2
;  4996          5107  2     MAP
;  4997          5108  2         TERM_LIST: REF VECTOR[,LONG];   ! Pointer to Terminator Table to use
;  4998          5109  2
;  4999          5110  2     LABEL
;  5000          5111  2         CHECK_THIS_TERMINATOR;          ! Label used to leave terminator
;  5001          5112  2                                         !      checking code
;  5002          5113  2
;  5003          5114  2     LOCAL
;  5004          5115  2         ACTION,                         ! Action index during number scanning
;  5005          5116  2         BACKUP_DIGIT_PTR,               ! Pointer to last good digit in number
;  5006          5117  2                                         !      --used to back up number scan
;  5007          5118  2         BACKUP_NUMBER_KIND,             ! Kind of numeric constant definite so
;  5008          5119  2                                         !      far--used to back up number scan
;  5009          5120  2         BEST_TOKEN_FOUND,               ! Pointer to the Operator Token Entry
;  5010          5121  2                                         !      for an operator with the right
;  5011          5122  2                                         !      name but the wrong kind
;  5012          5123  2         CLASS,                          ! Character class code of current char-
;  5013          5124  2                                         !      acter during number scanning
;  5014          5125  2         ENDPTR,                         ! Pointer to last char in an identifier
;  5015          5126  2         ERRORMSG,                       ! Error message condition code
;  5016          5127  2         INDEX,                          ! Index into look-up tables
;  5017          5128  2         NAMEPTR: REF VECTOR[,BYTE],     ! Pointer to name string in Percent Tbl
;  5018          5129  2         NEW_STARTPTR,                   ! Start pointer to lower case identifier
;  5019          5130  2         NUMBER_KIND,                    ! Kind of Numeric Constant Token found
;  5020          5131  2         PRID: REF PRID$ENTRY,           ! Pointer to Predefined Identifier Entry
;  5021          5132  2         QUOTE,                          ! Quote character which started the
;  5022          5133  2                                         !      current quoted string constant
;  5023          5134  2         STARTPTR: REF VECTOR[,BYTE],    ! Pointer to start of current token
;  5024          5135  2         STATE_INDEX,                    ! Current Number Scanner State Table
;  5025          5136  2                                         !      index during number scanning
;  5026          5137  2         TERMPTR: REF TOKEN$ENTRY,       ! Pointer to Terminator Toen Entry
;  5027          5138  2         TOKEN: REF TOKEN$ENTRY,         ! Pointer to Operator Token Entry
;  5028          5139  2         TOKENBUFFER: VECTOR[256,BYTE],  ! Vector in which current token is
;  5029          5140  2                                         !      accumulated as Counted ASCII
;  5030          5141  2         TOKEN_TYPE,                     ! Token's type; eg. TOKEN$K_STRING
;  5031          5142  2         TOKENLEN;                       ! The character length of current token
;  5032          5143  2
;  5033          5144  2
;  5034          5145  2
;  5035          5146  2     ! Start by scanning past any leading blanks.  Then mark the start location
;  5036          5147  2     ! of the token to be picked up.
```

```
;  5037     5148  2     !
;  5038     5149  2     WHILE .CHARTBL[.CHARPTR[0], CHRTBL$V_SPACE] DO
;  5039     5150  2         CHARPTR = .CHARPTR + 1;
;  5040     5151  2
;  5041     5152  2     STARTPTR = .CHARPTR;
;  5042     5153  2
;  5043     5154  2
;  5044     5155  2     ! If this took us to the carriage-return, return the terminator operator.
;  5045     5156  2     !
;  5046     5157  2     IF .CHARPTR[0] EQL CAR_RET
;  5047     5158  2     THEN
;  5048     5159  3         BEGIN
;  5049     5160  3         TERMINATOR_CODE = TOKEN$K_TERM_NONE;
;  5050     5161  3         TERMINATOR_LENGTH = 0;
;  5051     5162  3         RETURN TERMINATOR_TOKEN;
;  5052     5163  2         END;
;  5053     5164  2
;  5054     5165  2
;  5055     5166  2     ! Check for a valid Terminator Token.  The TERM_LIST parameter lists all
;  5056     5167  2     ! valid terminator tokens for the current expression.  If we have one of
;  5057     5168  2     ! those here, we set TERMINATOR_CODE appropriately and return the termi-
;  5058     5169  2     ! nator operator.
;  5059     5170  2     !
;  5060     5171  2     IF .CHARTBL[.CHARPTR[0], CHRTBL$V_TERMINATOR]
;  5061     5172  2     THEN
;  5062     5173  3         BEGIN
;  5063     5174  3
;  5064     5175  3
;  5065     5176  3         ! Loop over all allowed terminators in this context.
;  5066     5177  3         !
;  5067     5178  3         INCR I FROM 0 TO .TERM_LIST[-1] - 1 DO
;  5068     5179  4             BEGIN
;  5069     5180  4             TERMPTR = .TERM_LIST[.I] + TABLEBASE;
;  5070     5181  4
;  5071     5182  4
;  5072     5183  4             ! Check whether this terminator matches what we have in the input
;  5073     5184  4             ! line being scanned.
;  5074     5185  4             !
;  5075     5186  4             CHECK_THIS_TERMINATOR:
;  5076     5187  5                 BEGIN
;  5077     5188  5                 IF CH$NEQ(.TERMPTR[TOKEN$B_LENGTH], TERMPTR[TOKEN$A_NAME],
;  5078     5189  5                         .TERMPTR[TOKEN$B_LENGTH], .CHARPTR, 0)
;  5079     5190  5                 THEN
;  5080     5191  5                     LEAVE CHECK_THIS_TERMINATOR;
;  5081     5192  5
;  5082     5193  5                 IF .CHARTBL[.CHARPTR[0], CHRTBL$V_ALPHABETIC] AND
;  5083     5194  6                    (.CHARTBL[.CHARPTR[.TERMPTR[TOKEN$B_LENGTH]],
;  5084     5195  6                                                 CHRTBL$V_IDENT_MIDDLE] OR
;  5085     5196  6                     .CHARTBL[.CHARPTR[.TERMPTR[TOKEN$B_LENGTH]],
;  5086     5197  6                                                 CHRTBL$V_IDENT_END])
;  5087     5198  5                 THEN
;  5088     5199  5                     LEAVE CHECK_THIS_TERMINATOR;
;  5089     5200  5
;  5090     5201  6                 IF .TERMPTR[TOKEN$V_BALANCED_PARENS] AND (.PAREN_NESTING NEQ 0)
;  5091     5202  5                 THEN
;  5092     5203  5                     LEAVE CHECK_THIS_TERMINATOR;
;  5093     5204  5
```

```
;  5094     5205  5                IF .TERMPTR[TOKEN$V_MUST_BE_SINGLE] AND
;  5095     5206  6                   (.CHARPTR[0] EQL .CHARPTR[1])
;  5096     5207  5                THEN
;  5097     5208  5                    LEAVE CHECK_THIS_TERMINATOR;
;  5098     5209  5
;  5099     5210  5
;  5100     5211  5                ! This is a valid terminator in this context.  As a side-effect
;  5101     5212  5                ! save the terminator code in TERMINATOR_CODE and its length in
;  5102     5213  5                ! TERMINATOR_LENGTH.  Then return the terminator operator.
;  5103     5214  5                !
;  5104     5215  5                TERMINATOR_CODE = .TERMPTR[TOKEN$W_CODE];
;  5105     5216  5                TERMINATOR_LENGTH = .TERMPTR[TOKEN$B_LENGTH];
;  5106     5217  5                RETURN TERMINATOR_TOKEN;
;  5107     5218  5
;  5108     5219  4                END;                        ! End of CHECK_THIS_TERMINATOR block
;  5109     5220  4
;  5110     5221  3            END;                            ! End of INCR loop over terminators
;  5111     5222  3
;  5112     5223  2        END;                                ! End of terminator checking
;  5113     5224  2
;  5114     5225  2
;  5115     5226  2        ! Handle any language-specific special cases that must be sorted out before
;  5116     5227  2        ! we go through the normal lexical scanning code below.  Here we check for
;  5117     5228  2        ! those tokens that would be scanned incorrectly if we went through the
;  5118     5229  2        ! normal scanning mechanisms below.
;  5119     5230  2        !
;  5120     5231  2        IF .CHARTBL[.CHARPTR[0], CHRTBL$V_SPECIAL_CASE]
;  5121     5232  2        THEN
;  5122     5233  3            BEGIN
;  5123     5234  3            CASE .DBG$GB_LANGUAGE FROM DBG$K_MIN_LANGUAGE TO DBG$K_MAX_LANGUAGE OF
;  5124     5235  3                SET
;  5125     5236  3
;  5126     5237  3
;  5127     5238  3            ! Handle PL/I.  Here we special-case the -> operator.  We need to
;  5128     5239  3            ! mark "-" as being a single-character operator in the character
;  5129     5240  3            ! table so it does not get combined with itself or other operator
;  5130     5241  3            ! characters in expressions.  Hence we must special-case the one
;  5131     5242  3            ! situation where it can be combined, namely in "->".
;  5132     5243  3            !
;  5133     5244  3            [DBG$K_PLI]:
;  5134     5245  4                BEGIN
;  5135     5246  5                IF (.CHARPTR[0] EQL '-') AND (.CHARPTR[1] EQL '>')
;  5136     5247  4                THEN
;  5137     5248  5                    BEGIN
;  5138     5249  5                    CHARPTR = .CHARPTR + 2;
;  5139     5250  5                    RETURN PLI_ARROW_TOKEN;
;  5140     5251  4                    END;
;  5141     5252  4
;  5142     5253  3                END;
;  5143     5254  3
;  5144     5255  3
;  5145     5256  3            ! There are no special cases for any other language.  If we get
;  5146     5257  3            ! here, something is wrong (the code or character table is wrong.)
;  5147     5258  3            !
;  5148     5259  3            [INRANGE, OUTRANGE]:
;  5149     5260  3                $DBG_ERROR('DBGPARSER\LEXICAL_SCANNER 10');
;  5150     5261  3
```

```
;  5151          5262  3              TES;
;  5152          5263  3
;  5153          5264  2          END;
;  5154          5265  2
;  5155          5266  2
;  5156          5267  2          ! If the next token is one of the "special" symbols ".", "\", or "^",
;  5157          5268  2          ! and an operand is expected here, see if this token
;  5158          5269  2          ! could mean "current location", "current value", or "previous
;  5159          5270  2          ! location".  If so, return the appropriate Identifier Lexical Token Entry
;  5160          5271  2          ! but change the name to %CURLOC, %CURVAL, or %PREVLOC.
;  5161          5272  2          !
;  5162          5273  2          IF .OPERAND_EXPECTED AND .CHARTBL[.CHARPTR[0], CHRTBL$V_SPECIAL_SYMBOL]
;  5163          5274  2          THEN
;  5164          5275  3              BEGIN
;  5165          5276  3              CHARPTR = .CHARPTR + 1;
;  5166          5277  3              WHILE .CHARTBL[.CHARPTR[0], CHRTBL$V_SPACE] DO
;  5167          5278  3                  CHARPTR = .CHARPTR + 1;
;  5168          5279  3
;  5169          5280  3
;  5170          5281  3              ! Handle the special case where the user has said
;  5171          5282  3              ! SET BREAK . DO ( command-list). In this case we assume that
;  5172          5283  3              ! the "." means "current location".
;  5173          5284  3              !
;  5174          5285  3              IF .DBG$GB_SET_BREAK_FLAG
;  5175          5286  3              THEN
;  5176          5287  4                  BEGIN
;  5177          5288  4                  IF (.CHARPTR[0] EQL 'D') AND
;  5178          5289  4                     (.CHARPTR[1] EQL 'O') AND
;  5179          5290  5                     ((.CHARPTR[2] EQL ' ') OR (.CHARPTR[2] EQL '('))
;  5180          5291  4                  THEN
;  5181          5292  4                      RETURN CURLOC_TOKEN;
;  5182          5293  4                  IF (.CHARPTR[0] EQL 'W') AND
;  5183          5294  4                     (.CHARPTR[1] EQL 'H') AND
;  5184          5295  4                     (.CHARPTR[2] EQL 'E') AND
;  5185          5296  4                     (.CHARPTR[3] EQL 'N') AND
;  5186          5297  5                     ((.CHARPTR[4] EQL ' ') OR (.CHARPTR[4] EQL '('))
;  5187          5298  4                  THEN
;  5188          5299  4                      RETURN CURLOC_TOKEN;
;  5189          5300  3                  END;
;  5190          5301  3
;  5191          5302  3
;  5192          5303  3              ! We need to determine whether this special symbol is really one
;  5193          5304  3              ! of the debugger permanent symbols for previous, current, or next
;  5194          5305  3              ! location, or whether it is a language operator.
;  5195          5306  3              !
;  5196          5307  3              ! For "\", the only other possible meaning is as the start of a pathname.
;  5197          5308  3              ! (not the middle, since then "OPERAND_EXPECTED" would be false).
;  5198          5309  3              ! In that case, the next character must be the start of an identifier
;  5199          5310  3              ! or a "%" sign in %NAME. So in order to
;  5200          5311  3              ! intepret "\" as current value, we check here that the next token
;  5201          5312  3              ! is not the start of an identifier or a "%". (Note - in C, "%" is
;  5202          5313  3              ! an operator, so for example "\%3" means "current location mod 3",
;  5203          5314  3              ! so C is special-cased below).
;  5204          5315  3              !
;  5205          5316  3              ! "^" is an operator in some languages. The only one where it is
;  5206          5317  3              ! a prefix operator (the case that causes ambiguities)
;  5207          5318  3              ! is PLI, where it means "not". We resolve this by simply
```

```
;  5208          5319  3          ! assuming that "^" means prevloc if this is an address expression,
;  5209          5320  3          ! and means "not" in a language expression. This essentially
;  5210          5321  3          ! disallows "^" for prevloc in language expressions (%PREVLOC can
;  5211          5322  3          ! be used instead).
;  5212          5323  3          !
;  5213          5324  3          ! "." is highly overloaded. In many languages it can be the start
;  5214          5325  3          ! of a floating point number, so we check for the next character
;  5215          5326  3          ! being a digit here. It can also be the indirection operator
;  5216          5327  3          ! in an address expression. We check for the next character being
;  5217          5328  3          ! "(", "." or "\", and if so, assume that the dot means
;  5218          5329  3          ! indirection and not current location.
;  5219          5330  3          !
;  5220          5331  3          IF (NOT .CHARTBL[.CHARPTR[0], CHRTBL$V_IDENT_START]) AND
;  5221          5332  3             (NOT .CHARTBL[.CHARPTR[0], CHRTBL$V_DIGIT]) AND
;  5222          5333  4             (.CHARPTR[0] NEQ '%' OR
;  5223          5334  4             (.DBG$GB_LANGUAGE EQL DBG$K_C AND (NOT .ADDRESS_EXPRESSION)))
;  5224          5335  3          THEN
;  5225          5336  4              BEGIN
;  5226          5337  4              IF .STARTPTR[0] EQL '\' THEN RETURN CURVAL_TOKEN;
;  5227          5338  4              IF (.STARTPTR[0] EQL '^') AND
;  5228          5339  5                ((.DBG$GB_LANGUAGE NEQ DBG$K_PLI) OR .ADDRESS_EXPRESSION)
;  5229          5340  4              THEN
;  5230          5341  4                  RETURN PREVLOC_TOKEN;
;  5231          5342  4              IF .STARTPTR[0] EQL '.' AND
;  5232          5343  5                 NOT ((.CHARPTR[0] EQL '(') OR
;  5233          5344  5                      (.CHARPTR[0] EQL '.') OR
;  5234          5345  5                      (.CHARPTR[0] EQL '\'))
;  5235          5346  4              THEN
;  5236          5347  4                  RETURN CURLOC_TOKEN;
;  5237          5348  4
;  5238          5349  3              END;
;  5239          5350  3
;  5240          5351  3          CHARPTR = .STARTPTR;
;  5241          5352  2          END;
;  5242          5353  2
;  5243          5354  2
;  5244          5355  2          ! If we are expecting an operand next, check for the special DEBUG symbols
;  5245          5356  2          ! that begin with a percent sign "%".  This includes %LINE, %LABEL, %NAME,
;  5246          5357  2          ! and all the register names.
;  5247          5358  2          !
;  5248          5359  2          IF (.CHARPTR[0] EQL '%') AND
;  5249          5360  2             .OPERAND_EXPECTED     AND
;  5250          5361  2             .CHARTBL[.CHARPTR[1], CHRTBL$V_ALPHABETIC]
;  5251          5362  2          THEN
;  5252          5363  3              BEGIN
;  5253          5364  3
;  5254          5365  3
;  5255          5366  3              ! Accumulate the identifier after the "%"-sign.
;  5256          5367  3              !
;  5257          5368  3              CHARPTR = .CHARPTR + 1;
;  5258          5369  3              TOKENLEN = 1;
;  5259          5370  3              TOKENBUFFER[1] = '%';
;  5260          5371  3              WHILE .CHARTBL[.CHARPTR[0], CHRTBL$V_ALPHABETIC] OR
;  5261          5372  3                    .CHARTBL[.CHARPTR[0], CHRTBL$V_DIGIT]
;  5262          5373  3              DO
;  5263          5374  4                  BEGIN
;  5264          5375  4                  TOKENLEN = .TOKENLEN + 1;
```

```
;  5265          5376  4              TOKENBUFFER[.TOKENLEN] = .CHARPTR[0];
;  5266          5377  4              CHARPTR = .CHARPTR + 1;
;  5267          5378  3              END;
;  5268          5379  3
;  5269          5380  3          TOKENBUFFER[0] = .TOKENLEN;
;  5270          5381  3
;  5271          5382  3
;  5272          5383  3          ! Now look up the "%"-symbol in the DEBUG Percent Table.
;  5273          5384  3          !
;  5274          5385  3          INDEX = PERCENT_NOFIND;
;  5275          5386  3          INCR I FROM 0 TO .PERCENT_TABLE[-1] - 1 DO
;  5276          5387  4              BEGIN
;  5277          5388  4              NAMEPTR = .PERCENT_TABLE[.I] + TABLEBASE;
;  5278          5389  4              IF CH$EQL(.NAMEPTR[1], NAMEPTR[2], .TOKENLEN, TOKENBUFFER[1], 0)
;  5279          5390  4              THEN
;  5280          5391  5                  BEGIN
;  5281          5392  5                  INDEX = .NAMEPTR[0];
;  5282          5393  5                  EXITLOOP;
;  5283          5394  4                  END;
;  5284          5395  4
;  5285          5396  3              END;
;  5286          5397  3
;  5287          5398  3
;  5288          5399  3          ! Now do whatever further processing is appropriate for this "%"-symbol.
;  5289          5400  3          !
;  5290          5401  3          CASE .INDEX FROM PERCENT_NOFIND TO PERCENT_IDENT OF
;  5291          5402  3              SET
;  5292          5403  3
;  5293          5404  3
;  5294          5405  3              ! Handle the No-Find case.  We do not recognize this "%"-symbol,
;  5295          5406  3              ! so we do nothing.
;  5296          5407  3              !
;  5297          5408  3              [PERCENT_NOFIND]:
;  5298          5409  3                  0;
;  5299          5410  3
;  5300          5411  3
;  5301          5412  3              ! Handle %LINE and %LABEL.  Here we pick up the line or label
;  5302          5413  3              ! number that follows the keyword and construct an Identifier
;  5303          5414  3              ! Token Entry for the line or label and return that to the caller.
;  5304          5415  3              !
;  5305          5416  3              [PERCENT_LINE,
;  5306          5417  3               PERCENT_LABEL]:
;  5307          5418  4                  BEGIN
;  5308          5419  4
;  5309          5420  4
;  5310          5421  4                  ! Set up the fully spelled out keyword (%LINE or %LABEL) in
;  5311          5422  4                  ! TOKENBUFFER and set up the appropriate error message code.
;  5312          5423  4                  !
;  5313          5424  4                  IF .INDEX EQL PERCENT_LINE
;  5314          5425  4                  THEN
;  5315          5426  5                      BEGIN
;  5316          5427  5                      CH$MOVE(6, UPLIT BYTE(%ASCII '%LINE '), TOKENBUFFER[1]);
;  5317          5428  5                      TOKENLEN = 6;
;  5318          5429  5                      ERRORMSG = DBG$_SYNERRLINE;
;  5319          5430  5                      END
;  5320          5431  5
;  5321          5432  4                  ELSE
```

```
;  5322          5433  5          BEGIN
;  5323          5434  5          CH$MOVE(7, UPLIT BYTE(%ASCII '%LABEL '), TOKENBUFFER[1]);
;  5324          5435  5          TOKENLEN = 7;
;  5325          5436  5          ERRORMSG = DBG$_SYNERRLABEL;
;  5326          5437  4          END;
;  5327          5438  4
;  5328          5439  4
;  5329          5440  4      ! Check for and scan past the blanks after the %LINE or %LABEL
;  5330          5441  4      ! keyword and before the line or label number.
;  5331          5442  4      !
;  5332          5443  4      IF NOT .CHARTBL[.CHARPTR[0], CHRTBL$V_SPACE] THEN SIGNAL(.ERRORMSG);
;  5333          5444  4      WHILE .CHARTBL[.CHARPTR[0], CHRTBL$V_SPACE] DO
;  5334          5445  4          CHARPTR = .CHARPTR + 1;
;  5335          5446  4
;  5336          5447  4
;  5337          5448  4      ! Check that a digit follows.  Then strip off leading zeroes.
;  5338          5449  4      !
;  5339          5450  4      IF NOT .CHARTBL[.CHARPTR[0], CHRTBL$V_DIGIT]
;  5340          5451  4      THEN
;  5341          5452  4          SIGNAL(.ERRORMSG);
;  5342          5453  4
;  5343          5454  4      WHILE (.CHARPTR[0] EQL '0') AND
;  5344          5455  4            .CHARTBL[.CHARPTR[1], CHRTBL$V_DIGIT]
;  5345          5456  4      DO
;  5346          5457  4          CHARPTR = .CHARPTR + 1;
;  5347          5458  4
;  5348          5459  4
;  5349          5460  4      ! Now pick up the line or label number and copy all the digits
;  5350          5461  4      ! into TOKENBUFFER.
;  5351          5462  4      !
;  5352          5463  4      WHILE .CHARTBL[.CHARPTR[0], CHRTBL$V_DIGIT] DO
;  5353          5464  5          BEGIN
;  5354          5465  5          IF .TOKENLEN GEQ 255 THEN SIGNAL(.ERRORMSG);
;  5355          5466  5          TOKENLEN = .TOKENLEN + 1;
;  5356          5467  5          TOKENBUFFER[.TOKENLEN] = .CHARPTR[0];
;  5357          5468  5          CHARPTR = .CHARPTR + 1;
;  5358          5469  4          END;
;  5359          5470  4
;  5360          5471  4
;  5361          5472  4      ! If we are picking up a line number, we also allow a dot
;  5362          5473  4      ! followed by a statement number (e.g., %LINE 10.2).  Pick up
;  5363          5474  4      ! the statement number if present.
;  5364          5475  4      !
;  5365          5476  5      IF (.INDEX EQL PERCENT_LINE) AND (.CHARPTR[0] EQL '.')
;  5366          5477  4      THEN
;  5367          5478  5          BEGIN
;  5368          5479  5
;  5369          5480  5
;  5370          5481  5          ! Check for valid syntax and pick up the dot.
;  5371          5482  5          !
;  5372          5483  5          IF (NOT .CHARTBL[.CHARPTR[1], CHRTBL$V_DIGIT]) OR
;  5373          5484  6             (.TOKENLEN GEQ 255)
;  5374          5485  5          THEN
;  5375          5486  5              SIGNAL(.ERRORMSG);
;  5376          5487  5
;  5377          5488  5          TOKENLEN = .TOKENLEN + 1;
;  5378          5489  5          TOKENBUFFER[.TOKENLEN] = '.';
```

```
;  5379     5490  5                    CHARPTR = .CHARPTR + 1;
;  5380     5491  5
;  5381     5492  5
;  5382     5493  5                    ! Strip off leading zeroes.
;  5383     5494  5                    !
;  5384     5495  5                    WHILE (.CHARPTR[0] EQL '0') AND
;  5385     5496  5                          .CHARTBL[.CHARPTR[1], CHRTBL$V_DIGIT]
;  5386     5497  5                    DO
;  5387     5498  5                        CHARPTR = .CHARPTR + 1;
;  5388     5499  5
;  5389     5500  5
;  5390     5501  5                    ! Pick up the statement number itself.
;  5391     5502  5                    !
;  5392     5503  5                    WHILE .CHARTBL[.CHARPTR[0], CHRTBL$V_DIGIT] DO
;  5393     5504  6                        BEGIN
;  5394     5505  6                        IF .TOKENLEN GEQ 255 THEN SIGNAL(.ERRORMSG);
;  5395     5506  6                        TOKENLEN = .TOKENLEN + 1;
;  5396     5507  6                        TOKENBUFFER[.TOKENLEN] = .CHARPTR[0];
;  5397     5508  6                        CHARPTR = .CHARPTR + 1;
;  5398     5509  5                        END;
;  5399     5510  5
;  5400     5511  4                    END;
;  5401     5512  4
;  5402     5513  4
;  5403     5514  4                ! We now have a complete %LINE or %LABEL name string.  Create
;  5404     5515  4                ! and return an Identifier Token Entry for it.
;  5405     5516  4                !
;  5406     5517  4                TOKENBUFFER[0] = .TOKENLEN;
;  5407     5518  4                RETURN CREATE_OPERAND_TOKEN(TOKEN$K_IDENTIFIER, TOKENBUFFER);
;  5408     5519  4
;  5409     5520  3                END;                    ! End of %LINE and %LABEL processing
;  5410     5521  3
;  5411     5522  3
;  5412     5523  3                ! Handle %NAME.  Here we pick up the identifier symbol that follows
;  5413     5524  3                ! (directly or in quotes) and return an Identifier Token for it.
;  5414     5525  3                !
;  5415     5526  3                [PERCENT_NAME]:
;  5416     5527  4                    BEGIN
;  5417     5528  4
;  5418     5529  4
;  5419     5530  4                    ! Scan past any blanks after the %NAME keyword and before the
;  5420     5531  4                    ! actual name string itself.  Then set STARTPTR to that place.
;  5421     5532  4                    !
;  5422     5533  4                    WHILE .CHARTBL[.CHARPTR[0], CHRTBL$V_SPACE] DO
;  5423     5534  4                        CHARPTR = .CHARPTR + 1;
;  5424     5535  4
;  5425     5536  4                    STARTPTR = .CHARPTR;
;  5426     5537  4
;  5427     5538  4
;  5428     5539  4                    ! If the name string starts with a quote character, pick up all
;  5429     5540  4                    ! characters in the quoted string and move them to TOKENBUFFER.
;  5430     5541  4                    !
;  5431     5542  4                    IF .CHARTBL[.CHARPTR[0], CHRTBL$V_STRING_QUOTE]
;  5432     5543  4                    THEN
;  5433     5544  4                        SCAN_QUOTED_STRING(TOKENBUFFER, TOKEN_TYPE)
;  5434     5545  4
;  5435     5546  4
```

```
;  5436          5547  4              ! Otherwise, pick up any consecutive string of characters which
;  5437          5548  4              ! are allowed anywhere in identifiers.  For most languages,
;  5438          5549  4              ! this is any string composed of A - Z, 0 - 9, $, and _.
;  5439          5550  4              !
;  5440          5551  4              ELSE
;  5441          5552  5                  BEGIN
;  5442          5553  5                  WHILE .CHARTBL[.CHARPTR[0], CHRTBL$V_IDENT_START]  OR
;  5443          5554  5                        .CHARTBL[.CHARPTR[0], CHRTBL$V_IDENT_MIDDLE] OR
;  5444          5555  5                        .CHARTBL[.CHARPTR[0], CHRTBL$V_IDENT_END]
;  5445          5556  5                  DO
;  5446          5557  5                      CHARPTR = .CHARPTR + 1;
;  5447          5558  5
;  5448          5559  5                  TOKENLEN = .CHARPTR - .STARTPTR;
;  5449          5560  5                  IF .TOKENLEN EQL 0 THEN SIGNAL(DBG$_NAMSTRMIS);
;  5450          5561  5                  CH$MOVE(.TOKENLEN, .STARTPTR, TOKENBUFFER[1]);
;  5451          5562  5                  TOKENBUFFER[0] = .TOKENLEN;
;  5452          5563  4                  END;
;  5453          5564  4
;  5454          5565  4
;  5455          5566  4              ! Create and return an Identifier Token Entry for the symbol.
;  5456          5567  4              !
;  5457          5568  4              RETURN CREATE_OPERAND_TOKEN(TOKEN$K_IDENTIFIER, TOKENBUFFER);
;  5458          5569  4
;  5459          5570  3              END;                        ! End of %NAME processing
;  5460          5571  3
;  5461          5572  3
;  5462          5573  3          ! Handle "%"-symbol names that we recognize as identifiers.  This
;  5463          5574  3          ! includes all the register names.  Return an Identifier Token.
;  5464          5575  3          !
;  5465          5576  3          [PERCENT_IDENT]:
;  5466          5577  3              RETURN CREATE_OPERAND_TOKEN(TOKEN$K_IDENTIFIER, TOKENBUFFER);
;  5467          5578  3
;  5468          5579  3
;  5469          5580  3          ! Handle the %DEC operator.  This sets the radix to decimal for
;  5470          5581  3          ! the expression which constitutes its operand.
;  5471          5582  3          !
;  5472          5583  3          [PERCENT_DEC]:
;  5473          5584  3              RETURN RADIX_OP_DEC;
;  5474          5585  3
;  5475          5586  3
;  5476          5587  3          ! Handle the %HEX operator.  This sets the radix to hexadecimal for
;  5477          5588  3          ! the expression which constitutes its operand.
;  5478          5589  3          !
;  5479          5590  3          [PERCENT_HEX]:
;  5480          5591  3              RETURN RADIX_OP_HEX;
;  5481          5592  3
;  5482          5593  3
;  5483          5594  3          ! Handle the %OCT operator.  This sets the radix to octal for
;  5484          5595  3          ! the expression which constitutes its operand.
;  5485          5596  3          !
;  5486          5597  3          [PERCENT_OCT]:
;  5487          5598  3              RETURN RADIX_OP_OCT;
;  5488          5599  3
;  5489          5600  3
;  5490          5601  3          ! Handle the %BIN operator.  This sets the radix to binary for
;  5491          5602  3          ! the expression which constitutes its operand.
;  5492          5603  3          !
```

```
;  5493     5604  3            [PERCENT_BIN]:
;  5494     5605  3                RETURN RADIX_OP_BIN;
;  5495     5606  3
;  5496     5607  3
;  5497     5608  3            ! Any other CASE index is an internal DEBUG error.
;  5498     5609  3            !
;  5499     5610  3            [INRANGE, OUTRANGE]:
;  5500     5611  3                $DBG_ERROR('DBGPARSER\LEXICAL_SCANNER 20');
;  5501     5612  3
;  5502     5613  3            TES;
;  5503     5614  3
;  5504     5615  3
;  5505     5616  3        ! This is the end of the "%"-symbol processing.  If we have not recog-
;  5506     5617  3        ! nized the symbol yet, it is not a DEBUG special symbol so we reset
;  5507     5618  3        ! CHARPTR to point to the "%" sign again.  The hope is that we will
;  5508     5619  3        ! recognize it as a valid token later in the Lexical Scanner.
;  5509     5620  3        !
;  5510     5621  3        CHARPTR = .STARTPTR;
;  5511     5622  3
;  5512     5623  2        END;                                ! End of "%"-symbol scanning
;  5513     5624  2
;  5514     5625  2
;  5515     5626  2    ! See if this token is a quoted character string.  Quoted character strings
;  5516     5627  2    ! in this context are defined to start with a quote character (usually " or
;  5517     5628  2    ! '), to continue as a string of zero or more non-quote characters, and to
;  5518     5629  2    ! be terminated by a quote character.  The starting and ending quote char-
;  5519     5630  2    ! acters must be the same character and that quote character is represented
;  5520     5631  2    ! within the string by two consecutive quote characters.  If we find such
;  5521     5632  2    ! a quoted string, we accumulate it and return a String Constant Token.
;  5522     5633  2    !
;  5523     5634  2    IF .CHARTBL[.CHARPTR[0], CHRTBL$V_STRING_QUOTE]
;  5524     5635  2    THEN
;  5525     5636  3        BEGIN
;  5526     5637  3        SCAN_QUOTED_STRING(TOKENBUFFER, TOKEN_TYPE);
;  5527     5638  3        RETURN CREATE_OPERAND_TOKEN(.TOKEN_TYPE, TOKENBUFFER);
;  5528     5639  2        END;
;  5529     5640  2
;  5530     5641  2
;  5531     5642  2    ! See if this token is a numeric constant.  If so, we pick up the character
;  5532     5643  2    ! representation of the number and return it as a Numeric Constant Lexical
;  5533     5644  2    ! Token Entry.  This scan must be done before the Identifier scan below so
;  5534     5645  2    ! that COBOL integers get interpreted as numbers, not identifiers.
;  5535     5646  2    !
;  5536     5647  2    ! This code simulates a Finite-State Machine (FSM) which accepts any valid
;  5537     5648  2    ! numeric constant in the current language.  The machine is defined by a
;  5538     5649  2    ! state table where each state has a set of allowed transitions to other
;  5539     5650  2    ! states.  Each transition is selected by the next input character and has
;  5540     5651  2    ! an associated action routine defined below.  When a numeric constant has
;  5541     5652  2    ! been recognized, a transition is taken whose action routine builds and
;  5542     5653  2    ! returns a Lexical Token Entry for the numeric constant.
;  5543     5654  2    !
;  5544     5655  2    IF .CHARTBL[.CHARPTR[0], CHRTBL$V_NUMBER_START] AND
;  5545     5656  3       ((NOT .ADDRESS_EXPRESSION) OR (.CHARPTR[0] NEQ '.'))
;  5546     5657  2    THEN
;  5547     5658  3        BEGIN
;  5548     5659  3
;  5549     5660  3
```

```
;  5550     5661  3          ! Start at the Start State for the Finite-State Machine that will scan
;  5551     5662  3          ! the number according to the rules of the current language.  This
;  5552     5663  3          ! start state is ordinarily zero. If the language is ADA and the radix
;  5553     5664  3          ! is not decimal, however, we start at the B_START_STATE, which is
;  5554     5665  3          ! the entry to the part of the ADA number scanner that picks up
;  5555     5666  3          ! based numbers.  Also, set up the initial number kind based on the
;  5556     5667  3          ! current radix setting assuming it will be an integer.
;  5557     5668  3          !
;  5558     5669  3          STATE_INDEX = 0;
;  5559     5670  3          IF (.DBG$GB_LANGUAGE EQL DBG$K_ADA) AND
;  5560     5671  4             (.EXPRESSION_RADIX NEQ DBG$K_DECIMAL)
;  5561     5672  3          THEN
;  5562     5673  3              STATE_INDEX = REMEMBER_ADA_B_START_STATE;
;  5563     5674  3
;  5564     5675  3          NUMBER_KIND = TOKEN$K_INTEGER;
;  5565     5676  3          IF .EXPRESSION_RADIX EQL DBG$K_HEX
;  5566     5677  3          THEN
;  5567     5678  3              NUMBER_KIND = TOKEN$K_HEX_INTEGER;
;  5568     5679  3
;  5569     5680  3          IF .EXPRESSION_RADIX EQL DBG$K_OCTAL
;  5570     5681  3          THEN
;  5571     5682  3              NUMBER_KIND = TOKEN$K_OCT_INTEGER;
;  5572     5683  3
;  5573     5684  3          IF .EXPRESSION_RADIX EQL DBG$K_BINARY
;  5574     5685  3          THEN
;  5575     5686  3              NUMBER_KIND = TOKEN$K_BIN_INTEGER;
;  5576     5687  3
;  5577     5688  3          BACKUP_NUMBER_KIND = .NUMBER_KIND;
;  5578     5689  3
;  5579     5690  3
;  5580     5691  3          ! Then loop through the machine states, selecting each next state based
;  5581     5692  3          ! on the next input character and performing the appropriate action for
;  5582     5693  3          ! each transition, until the whole number is picked up.
;  5583     5694  3          !
;  5584     5695  3          WHILE TRUE DO
;  5585     5696  4              BEGIN
;  5586     5697  4
;  5587     5698  4
;  5588     5699  4              ! Get the character class code for the next input character.
;  5589     5700  4              ! Small kludge for ADA: In X[1..5], we want to treat ".."
;  5590     5701  4              ! as terminating the "1", not as part of the number.
;  5591     5702  4              !
;  5592     5703  4              CLASS = .CHARTBL[.CHARPTR[0], CHRTBL$V_NUMBER_CLASS];
;  5593     5704  4              IF .DBG$GB_LANGUAGE EQL DBG$K_ADA
;  5594     5705  4              THEN
;  5595     5706  4                  IF .CHARPTR[0] EQL '.' AND .CHARPTR[1] EQL '.'
;  5596     5707  4                  THEN
;  5597     5708  4                      CLASS = NUMST$K_CLASS_OTHER;
;  5598     5709  4
;  5599     5710  4
;  5600     5711  4              ! Loop through the transitions from this state until we find a
;  5601     5712  4              ! transition for this character class or for NUMST$K_CLASS_OTHER
;  5602     5713  4              ! (the class of all other characters).  Pick up the action index
;  5603     5714  4              ! and next state associated with this transition of the Finite-
;  5604     5715  4              ! State Machine.
;  5605     5716  4              !
;  5606     5717  5              WHILE (.STATE_TABLE[.STATE_INDEX, NUMST$B_CHAR_CLASS] NEQ
```

```
;  5607     5718  4                                        NUMST$K_CLASS_OTHER) AND
;  5608     5719  5                 (.STATE_TABLE[.STATE_INDEX, NUMST$B_CHAR_CLASS] NEQ .CLASS)
;  5609     5720  4             DO
;  5610     5721  4                 STATE_INDEX = .STATE_INDEX + 1;
;  5611     5722  4
;  5612     5723  4             ACTION = .STATE_TABLE[.STATE_INDEX, NUMST$B_ACTION];
;  5613     5724  4             STATE_INDEX = .STATE_TABLE[.STATE_INDEX, NUMST$W_NEXT_STATE];
;  5614     5725  4
;  5615     5726  4
;  5616     5727  4             ! Case on the action code to select the appropriate semantic
;  5617     5728  4             ! action routine for this state transition.
;  5618     5729  4             !
;  5619     5730  4             CASE .ACTION FROM NUMST$K_MIN_ACTION TO NUMST$K_MAX_ACTION OF
;  5620     5731  4                 SET
;  5621     5732  4
;  5622     5733  4
;  5623     5734  4                 ! Some state transitions require no semantic action, so we
;  5624     5735  4                 ! provide a do-nothing action routine.
;  5625     5736  4                 !
;  5626     5737  4                 [NUMST$K_ACT_DO_NOTHING]:
;  5627     5738  4                     0;
;  5628     5739  4
;  5629     5740  4
;  5630     5741  4                 ! Go past a digit in the integer part of the number.  Set the
;  5631     5742  4                 ! digit backup pointer to the last good digit seen (this one).
;  5632     5743  4                 !
;  5633     5744  4                 [NUMST$K_ACT_GO_PAST_DIGIT]:
;  5634     5745  4                     BACKUP_DIGIT_PTR = .CHARPTR;
;  5635     5746  4
;  5636     5747  4
;  5637     5748  4                 ! At the decimal point, mark that we have a floating-point
;  5638     5749  4                 ! number.
;  5639     5750  4                 !
;  5640     5751  4                 [NUMST$K_ACT_MARK_DEC_PT]:
;  5641     5752  4                     NUMBER_KIND = TOKEN$K_FLOATING;
;  5642     5753  4
;  5643     5754  4
;  5644     5755  4                 ! At a fraction digit, set the backup pointer to the last good
;  5645     5756  4                 ! digit seen (i.e., this one) and note that we really have a
;  5646     5757  4                 ! floating-point number even if we must back up the scan.
;  5647     5758  4                 !
;  5648     5759  4                 [NUMST$K_ACT_GO_PAST_FRAC]:
;  5649     5760  5                     BEGIN
;  5650     5761  5                     BACKUP_DIGIT_PTR = .CHARPTR;
;  5651     5762  5                     BACKUP_NUMBER_KIND = TOKEN$K_FLOATING;
;  5652     5763  4                     END;
;  5653     5764  4
;  5654     5765  4
;  5655     5766  4                 ! Mark that we found an E exponent marker.
;  5656     5767  4                 !
;  5657     5768  4                 [NUMST$K_ACT_MARK_E_EXP]:
;  5658     5769  4                     NUMBER_KIND = TOKEN$K_EXP_E_FLOAT;
;  5659     5770  4
;  5660     5771  4
;  5661     5772  4                 ! Mark that we found a D exponent marker.
;  5662     5773  4                 !
;  5663     5774  4                 [NUMST$K_ACT_MARK_D_EXP]:
```

```
;  5664          5775  4                    NUMBER_KIND = TOKEN$K_EXP_D_FLOAT;
;  5665          5776  4
;  5666          5777  4
;  5667          5778  4                ! Mark that we found a G exponent marker.
;  5668          5779  4                !
;  5669          5780  4                [NUMST$K_ACT_MARK_G_EXP]:
;  5670          5781  4                    NUMBER_KIND = TOKEN$K_EXP_G_FLOAT;
;  5671          5782  4
;  5672          5783  4
;  5673          5784  4                ! Mark that we found a Q exponent marker.
;  5674          5785  4                !
;  5675          5786  4                [NUMST$K_ACT_MARK_Q_EXP]:
;  5676          5787  4                    NUMBER_KIND = TOKEN$K_EXP_Q_FLOAT;
;  5677          5788  4
;  5678          5789  4
;  5679          5790  4                ! At a pack decimal digit, set the backup pointer to the last good
;  5680          5791  4                ! digit seen (i.e., this one) and note that we really have a
;  5681          5792  4                ! pack decimal number even if we must back up the scan.
;  5682          5793  4                !
;  5683          5794  4                [NUMST$K_ACT_GO_PAST_PACK]:
;  5684          5795  5                    BEGIN
;  5685          5796  5                    BACKUP_DIGIT_PTR = .CHARPTR;
;  5686          5797  6                    IF NOT (.NUMBER_KIND EQL TOKEN$K_HEX_INTEGER OR
;  5687          5798  6                            .NUMBER_KIND EQL TOKEN$K_OCT_INTEGER OR
;  5688          5799  6                            .NUMBER_KIND EQL TOKEN$K_BIN_INTEGER)
;  5689          5800  5                    THEN
;  5690          5801  5                        BACKUP_NUMBER_KIND = TOKEN$K_PACK_DECIMAL;
;  5691          5802  4                    END;
;  5692          5803  4
;  5693          5804  4
;  5694          5805  4                ! At a pack decimal digit, set the backup pointer to the last good
;  5695          5806  4                ! digit seen (i.e., this one) and note that we really have a
;  5696          5807  4                ! pack decimal number even if we must back up the scan.
;  5697          5808  4                !
;  5698          5809  4                [NUMST$K_ACT_GO_PAST_PACK_FRAC]:
;  5699          5810  5                    BEGIN
;  5700          5811  5                    BACKUP_DIGIT_PTR = .CHARPTR;
;  5701          5812  6                    IF NOT (.NUMBER_KIND EQL TOKEN$K_HEX_INTEGER OR
;  5702          5813  6                            .NUMBER_KIND EQL TOKEN$K_OCT_INTEGER OR
;  5703          5814  6                            .NUMBER_KIND EQL TOKEN$K_BIN_INTEGER)
;  5704          5815  5                    THEN
;  5705          5816  5                        BACKUP_NUMBER_KIND = TOKEN$K_PACK_DECIMAL
;  5706          5817  5                    ELSE
;  5707          5818  5                        BACKUP_NUMBER_KIND = TOKEN$K_FLOATING;
;  5708          5819  5
;  5709          5820  4                    END;
;  5710          5821  4
;  5711          5822  4
;  5712          5823  4                ! We have a number but we scanned too far.  This can happen in
;  5713          5824  4                ! FORTRAN with "25.EQ." where we pick up the "." and "E" since
;  5714          5825  4                ! that could be part of a number.  Back up the scan pointer to
;  5715          5826  4                ! the true end of the number and return the appropriate Numeric
;  5716          5827  4                ! Constant Lexical Token Entry.
;  5717          5828  4                !
;  5718          5829  4                [NUMST$K_ACT_BACKUP_PTRS]:
;  5719          5830  5                    BEGIN
;  5720          5831  5                    NUMBER_KIND = .BACKUP_NUMBER_KIND;
```

```
;  5721          5832  5                CHARPTR = .BACKUP_DIGIT_PTR + 1;
;  5722          5833  5                TOKENLEN = .CHARPTR - .STARTPTR;
;  5723          5834  5                IF .TOKENLEN GTR 255 THEN SIGNAL(DBG$_NUMCONLONG);
;  5724          5835  5                TOKENBUFFER[0] = .TOKENLEN;
;  5725          5836  5                CH$MOVE(.TOKENLEN, .STARTPTR, TOKENBUFFER[1]);
;  5726          5837  5                RETURN CREATE_OPERAND_TOKEN(.NUMBER_KIND, TOKENBUFFER);
;  5727          5838  4                END;
;  5728          5839  4
;  5729          5840  4
;  5730          5841  4            ! We have a complete and valid numeric constant.  Create a
;  5731          5842  4            ! Numeric Constant Lexical Token Entry for it and return a
;  5732          5843  4            ! pointer to that Token Entry to the caller.
;  5733          5844  4            !
;  5734          5845  4            [NUMST$K_ACT_GOT_NUMBER]:
;  5735          5846  5                BEGIN
;  5736          5847  5                TOKENLEN = .CHARPTR - .STARTPTR;
;  5737          5848  5                IF .TOKENLEN GTR 255 THEN SIGNAL(DBG$_NUMCONLONG);
;  5738          5849  5                TOKENBUFFER[0] = .TOKENLEN;
;  5739          5850  5                CH$MOVE(.TOKENLEN, .STARTPTR, TOKENBUFFER[1]);
;  5740          5851  5                RETURN CREATE_OPERAND_TOKEN(.NUMBER_KIND, TOKENBUFFER);
;  5741          5852  4                END;
;  5742          5853  4
;  5743          5854  4
;  5744          5855  4            ! We have a complete and valid pack decimal constant.  Create a
;  5745          5856  4            ! Numeric Constant Lexical Token Entry for it and return a
;  5746          5857  4            ! pointer to that Token Entry to the caller.
;  5747          5858  4            !
;  5748          5859  4            [NUMST$K_ACT_GOT_PACK_NUMBER]:
;  5749          5860  5                BEGIN
;  5750          5861  6                IF NOT (.NUMBER_KIND EQL TOKEN$K_HEX_INTEGER OR
;  5751          5862  6                        .NUMBER_KIND EQL TOKEN$K_OCT_INTEGER OR
;  5752          5863  6                        .NUMBER_KIND EQL TOKEN$K_BIN_INTEGER)
;  5753          5864  5                THEN
;  5754          5865  5                    NUMBER_KIND = TOKEN$K_PACK_DECIMAL;
;  5755          5866  5
;  5756          5867  5                TOKENLEN = .CHARPTR - .STARTPTR;
;  5757          5868  5                IF .TOKENLEN GTR 255 THEN SIGNAL(DBG$_NUMCONLONG);
;  5758          5869  5                TOKENBUFFER[0] = .TOKENLEN;
;  5759          5870  5                CH$MOVE(.TOKENLEN, .STARTPTR, TOKENBUFFER[1]);
;  5760          5871  5                RETURN CREATE_OPERAND_TOKEN(.NUMBER_KIND, TOKENBUFFER);
;  5761          5872  4                END;
;  5762          5873  4
;  5763          5874  4
;  5764          5875  4            ! This is not a valid numeric constant.  Back up CHARPTR to the
;  5765          5876  4            ! start of the lexical token and exit the Number Scanner code.
;  5766          5877  4            !
;  5767          5878  4            [NUMST$K_ACT_NOT_NUMBER]:
;  5768          5879  5                BEGIN
;  5769          5880  5                CHARPTR = .STARTPTR;
;  5770          5881  5                EXITLOOP;
;  5771          5882  4                END;
;  5772          5883  4
;  5773          5884  4
;  5774          5885  4            ! In cobol, 12A will be a name if the mode setting is not hex.
;  5775          5886  4            !
;  5776          5887  4            [NUMST$K_ACT_COB_CKHEX]:
;  5777          5888  5                BEGIN
```

```
;  5778     5889  5                IF .EXPRESSION_RADIX NEQ DBG$K_HEX
;  5779     5890  5                THEN
;  5780     5891  6                    BEGIN
;  5781     5892  6                    CHARPTR = .STARTPTR;
;  5782     5893  6                    EXITLOOP;
;  5783     5894  5                    END;
;  5784     5895  5
;  5785     5896  4                END;
;  5786     5897  4
;  5787     5898  4
;  5788     5899  4            ! In COBOL, a number such as 123 can be a name.  Such a name
;  5789     5900  4            ! must be entered as %NAME 123.  However, if we find a number
;  5790     5901  4            ! followed by a valid identifier character, as in 123A or 12-3,
;  5791     5902  4            ! then we actually have an identifier, so we exit the number
;  5792     5903  4            ! scanning loop without returning a number token.  If the
;  5793     5904  4            ! number ended with any other character, we return a valid
;  5794     5905  4            ! number token.
;  5795     5906  4            !
;  5796     5907  4            [NUMST$K_ACT_COB_CKNUM]:
;  5797     5908  5                BEGIN
;  5798     5909  5                IF .CHARTBL[.CHARPTR[0], CHRTBL$V_IDENT_START]  OR
;  5799     5910  5                   .CHARTBL[.CHARPTR[0], CHRTBL$V_IDENT_MIDDLE] OR
;  5800     5911  5                   .CHARTBL[.CHARPTR[0], CHRTBL$V_IDENT_END]
;  5801     5912  5                THEN
;  5802     5913  6                    BEGIN
;  5803     5914  6                    CHARPTR = .STARTPTR;
;  5804     5915  6                    EXITLOOP;
;  5805     5916  5                    END;
;  5806     5917  5
;  5807     5918  6                IF NOT ((.NUMBER_KIND EQL TOKEN$K_HEX_INTEGER) OR
;  5808     5919  6                        (.NUMBER_KIND EQL TOKEN$K_OCT_INTEGER) OR
;  5809     5920  6                        (.NUMBER_KIND EQL TOKEN$K_BIN_INTEGER))
;  5810     5921  5                THEN
;  5811     5922  5                    NUMBER_KIND = TOKEN$K_PACK_DECIMAL;
;  5812     5923  5
;  5813     5924  5                TOKENLEN = .CHARPTR - .STARTPTR;
;  5814     5925  5                IF .TOKENLEN GTR 255 THEN SIGNAL(DBG$_NUMCONLONG);
;  5815     5926  5                TOKENBUFFER[0] = .TOKENLEN;
;  5816     5927  5                CH$MOVE(.TOKENLEN, .STARTPTR, TOKENBUFFER[1]);
;  5817     5928  5                RETURN CREATE_OPERAND_TOKEN(.NUMBER_KIND, TOKENBUFFER);
;  5818     5929  4                END;
;  5819     5930  4
;  5820     5931  4
;  5821     5932  4            ! Save the base of a number in ADA, where a number can be
;  5822     5933  4            ! of the form base#number.
;  5823     5934  4            !
;  5824     5935  4            [NUMST$K_ACT_SAVE_BASE]:
;  5825     5936  4                0;  !<<<----------------------------
;  5826     5937  4
;  5827     5938  4
;  5828     5939  4            ! Any other action index constitutes an internal error.
;  5829     5940  4            !
;  5830     5941  4            [NUMST$K_ACT_GIVE_ERROR,
;  5831     5942  4             INRANGE,
;  5832     5943  4             OUTRANGE]:
;  5833     5944  4                $DBG_ERROR('DBGPARSER\LEXICAL_SCANNER 30');
;  5834     5945  4
```

```
;  5835          5946  4                TES;
;  5836          5947  4
;  5837          5948  4
;  5838          5949  4            ! Go on to the next character in the buffer and loop.
;  5839          5950  4            !
;  5840          5951  4            CHARPTR = .CHARPTR + 1;
;  5841          5952  4
;  5842          5953  3            END;                            ! End of WHILE loop over number states
;  5843          5954  3
;  5844          5955  2        END;                                ! End of numeric constant scanning
;  5845          5956  2
;  5846          5957  2
;  5847          5958  2        ! See if this token is an identifier according to the rules of the current
;  5848          5959  2        ! language.  If so, pick up the whole identifier, see if this identifier is
;  5849          5960  2        ! actually an operator (such as AND or MOD) in the current language, and
;  5850          5961  2        ! return either an Identifier Token or an Operator Token.
;  5851          5962  2        !
;  5852          5963  2        IF .CHARTBL[.CHARPTR[0], CHRTBL$V_IDENT_START]
;  5853          5964  2        THEN
;  5854          5965  3            BEGIN
;  5855          5966  3
;  5856          5967  3
;  5857          5968  3            ! We have a valid start character for an identifier in the current
;  5858          5969  3            ! language.  Now scan through the identifier as long as we have valid
;  5859          5970  3            ! middle characters and set ENDPTR each time we find a valid end char-
;  5860          5971  3            ! acter for an identifier.  At the end of the scan we set CHARPTR to
;  5861          5972  3            ! point to the first character after the identifier end character.
;  5862          5973  3            !
;  5863          5974  3            ENDPTR = .CHARPTR - 1;
;  5864          5975  3            WHILE TRUE DO
;  5865          5976  4                BEGIN
;  5866          5977  4                IF .CHARTBL[.CHARPTR[0], CHRTBL$V_IDENT_END]
;  5867          5978  4                THEN
;  5868          5979  5                    BEGIN
;  5869          5980  5                    ENDPTR = .CHARPTR;
;  5870          5981  5                    IF NOT .CHARTBL[.CHARPTR[0], CHRTBL$V_IDENT_MIDDLE] THEN EXITLOOP;
;  5871          5982  4                    END;
;  5872          5983  4
;  5873          5984  4                CHARPTR = .CHARPTR + 1;
;  5874          5985  4                IF (NOT .CHARTBL[.CHARPTR[0], CHRTBL$V_IDENT_MIDDLE]) AND
;  5875          5986  5                   (NOT .CHARTBL[.CHARPTR[0], CHRTBL$V_IDENT_END])
;  5876          5987  4                   THEN EXITLOOP;
;  5877          5988  3                END;
;  5878          5989  3
;  5879          5990  3            CHARPTR = .ENDPTR + 1;
;  5880          5991  3
;  5881          5992  3
;  5882          5993  3            ! Copy the identifier name to TOKENBUFFER.
;  5883          5994  3            !
;  5884          5995  3            TOKENLEN = .CHARPTR - .STARTPTR;
;  5885          5996  3            IF .TOKENLEN GTR 255 THEN SIGNAL(DBG$_IDENTLONG);
;  5886          5997  3            CH$MOVE(.TOKENLEN, .STARTPTR, TOKENBUFFER[1]);
;  5887          5998  3            TOKENBUFFER[0] = .TOKENLEN;
;  5888          5999  3
;  5889          6000  3
;  5890          6001  3            ! If the language is C, then we want to make sure that the identifier
;  5891          6002  3            ! preserves the original casing (upper/lower). This is because
```

```
; 5892          6003  3          ! in C, upper case XXX is a different variable from lower case xxx,
; 5893          6004  3          ! for example. So we copy characters from the original command
; 5894          6005  3          ! buffer instead of the up-cased command buffer.
; 5895          6006  3          ! We use a flag "CASING_SIGNIFICANT" which is set in the SET_LANGUAGE
; 5896          6007  3          ! routine for this. At present, C is the only language that sets
; 5897          6008  3          ! this flag to TRUE.
; 5898          6009  3          !
; 5899          6010  3          IF .CASING_SIGNIFICANT
; 5900          6011  3          THEN
; 5901          6012  4              BEGIN
; 5902          6013  4              IF (.STARTPTR LSS .DBG$GL_UPCASE_COMMAND_PTR[0]) OR
; 5903          6014  5                 (.STARTPTR GTR .DBG$GL_UPCASE_COMMAND_PTR[1])
; 5904          6015  4              THEN
; 5905          6016  4                  $DBG_ERROR('DBGPARSER\DBG$LEXICAL_SCANNER 40');
; 5906          6017  4
; 5907          6018  4              NEW_STARTPTR = (.STARTPTR - .DBG$GL_UPCASE_COMMAND_PTR[0]) +
; 5908          6019  4                             .DBG$GL_ORIG_COMMAND_PTR;
; 5909          6020  4              CH$MOVE(.TOKENLEN, .NEW_STARTPTR, TOKENBUFFER[1]);
; 5910          6021  4              END;
; 5911          6022  3
; 5912          6023  3
; 5913          6024  3          ! See if this identifier is actually the name of an operator.  We scan
; 5914          6025  3          ! a language-specific operator table to determine this.  If so, return
; 5915          6026  3          ! the corresponding Operator Token.
; 5916          6027  3          !
; 5917          6028  3          BEST_TOKEN_FOUND = 0;
; 5918          6029  3          INCR I FROM 0 TO .IDENT_OPERATOR_TABLE[-1] - 1 DO
; 5919          6030  4              BEGIN
; 5920          6031  4              TOKEN = .IDENT_OPERATOR_TABLE[.I] + TABLEBASE;
; 5921          6032  4              IF CH$EQL(.TOKEN[TOKEN$B_OPLEN],
; 5922          6033  4                  TOKEN[TOKEN$A_OPNAME], .TOKENLEN, TOKENBUFFER[1], 0)
; 5923          6034  4              THEN
; 5924          6035  5                  BEGIN
; 5925          6036  6                  IF (.OPERAND_EXPECTED AND
; 5926          6037  5                          (.TOKEN[TOKEN$B_KIND] EQL TOKEN$K_PREFIX_OP)) OR
; 5927          6038  6                     ((NOT .OPERAND_EXPECTED) AND
; 5928          6039  6                          (.TOKEN[TOKEN$B_KIND] NEQ TOKEN$K_PREFIX_OP))
; 5929          6040  5                  THEN
; 5930          6041  5                      RETURN .TOKEN
; 5931          6042  5
; 5932          6043  5                  ELSE
; 5933          6044  5                      BEST_TOKEN_FOUND = .TOKEN;
; 5934          6045  5
; 5935          6046  4                  END;
; 5936          6047  4
; 5937          6048  3              END;
; 5938          6049  3
; 5939          6050  3          IF .BEST_TOKEN_FOUND NEQ 0 THEN RETURN .BEST_TOKEN_FOUND;
; 5940          6051  3
; 5941          6052  3
; 5942          6053  3          ! See if this identifier is the name of a built-in function.  We scan
; 5943          6054  3          ! a language-specific built-in function table to determine this.  If
; 5944          6055  3          ! so, return the corresponding Operand Token.
; 5945          6056  3          !
; 5946          6057  3          BEST_TOKEN_FOUND = 0;
; 5947          6058  3          INCR I FROM 0 TO .BIF_TABLE[-1] - 1 DO
; 5948          6059  4              BEGIN
```

```
;  5949          6060  4              TOKEN = .BIF_TABLE[.I] + TABLEBASE;
;  5950          6061  4              IF CH$EQL(.TOKEN[TOKEN$B_LENGTH], TOKEN[TOKEN$A_NAME],
;  5951          6062  4                        .TOKENLEN, TOKENBUFFER[1], 0)
;  5952          6063  4              THEN
;  5953          6064  4                  IF .OPERAND_EXPECTED
;  5954          6065  4                  THEN
;  5955          6066  4                      RETURN .TOKEN
;  5956          6067  4                  ELSE
;  5957          6068  4                      BEST_TOKEN_FOUND = .TOKEN;
;  5958          6069  3              END;
;  5959          6070  3
;  5960          6071  3          IF .BEST_TOKEN_FOUND NEQ 0 THEN RETURN .BEST_TOKEN_FOUND;
;  5961          6072  3
;  5962          6073  3
;  5963          6074  3          ! It is not an operator or a built-in function.  Hence we return an
;  5964          6075  3          ! identifier Token for the symbol.  Note that we do not accept a
;  5965          6076  3          ! zero-length identifier (one that has no valid end character).
;  5966          6077  3          !
;  5967          6078  3          IF .TOKENLEN NEQ 0
;  5968          6079  3          THEN
;  5969          6080  3              RETURN CREATE_OPERAND_TOKEN(TOKEN$K_IDENTIFIER, TOKENBUFFER);
;  5970          6081  3
;  5971          6082  2          END;                                ! End of identifier scanning
;  5972          6083  2
;  5973          6084  2
;  5974          6085  2      ! See if this token is an operator symbol (such as +, *, /, or :=) in the
;  5975          6086  2      ! current language.  If so, return an Operator Token to the caller.
;  5976          6087  2      !
;  5977          6088  2      IF .CHARTBL[.CHARPTR[0], CHRTBL$V_OPCHAR]
;  5978          6089  2      THEN
;  5979          6090  3          BEGIN
;  5980          6091  3
;  5981          6092  3
;  5982          6093  3          ! Determine where the operator symbol ends.  By classifying the avail-
;  5983          6094  3          ! able operator characters as prefix, infix, or postfix characters, we
;  5984          6095  3          ! can ensure that we break operator character strings apart at postfix-
;  5985          6096  3          ! infix, infix-prefix, and postfix-prefix boundaries.  An operator char-
;  5986          6097  3          ! acter which does not fall in any of those classes cannot be combined
;  5987          6098  3          ! with other characters and is always a symbol by itself (such as "(").
;  5988          6099  3          !
;  5989          6100  3          WHILE .CHARTBL[.CHARPTR[0], CHRTBL$V_OPCHAR_PREFIX] DO
;  5990          6101  3              CHARPTR = .CHARPTR + 1;
;  5991          6102  3
;  5992          6103  3          WHILE .CHARTBL[.CHARPTR[0], CHRTBL$V_OPCHAR_INFIX] DO
;  5993          6104  3              CHARPTR = .CHARPTR + 1;
;  5994          6105  3
;  5995          6106  3          WHILE .CHARTBL[.CHARPTR[0], CHRTBL$V_OPCHAR_POSTFIX] DO
;  5996          6107  3              CHARPTR = .CHARPTR + 1;
;  5997          6108  3
;  5998          6109  3          IF .CHARPTR EQLU .STARTPTR THEN CHARPTR = .CHARPTR + 1;
;  5999          6110  3
;  6000          6111  3
;  6001          6112  3          ! Copy the operator symbol to TOKENBUFFER.
;  6002          6113  3          !
;  6003          6114  3          TOKENLEN = .CHARPTR - .STARTPTR;
;  6004          6115  3          IF .TOKENLEN GTR 255 THEN SIGNAL(DBG$_INVOPSYM);
;  6005          6116  3          TOKENBUFFER[0] = .TOKENLEN;
```

```
;  6006          6117  3          CH$MOVE(.TOKENLEN, .STARTPTR, TOKENBUFFER[1]);
;  6007          6118  3
;  6008          6119  3
;  6009          6120  3          ! We now have an operator symbol in TOKENBUFFER.  Look it up in the
;  6010          6121  3          ! operator table for the current language, and if found, return the
;  6011          6122  3          ! corresponding Operator Token.  Note that we only accept an operator
;  6012          6123  3          ! if it is a Primary operator (which we accept in both language and
;  6013          6124  3          ! address expressions) or if we are in a language expression.
;  6014          6125  3          !
;  6015          6126  3          BEST_TOKEN_FOUND = 0;
;  6016          6127  3          INCR I FROM 0 TO .OPCHAR_OPERATOR_TABLE[-1] - 1 DO
;  6017          6128  4              BEGIN
;  6018          6129  4              TOKEN = .OPCHAR_OPERATOR_TABLE[.I] + TABLEBASE;
;  6019          6130  4              IF (.TOKEN[TOKEN$V_PRIMARY] OR (NOT .ADDRESS_EXPRESSION)) AND
;  6020          6131  4                 CH$EQL(.TOKEN[TOKEN$B_OPLEN],
;  6021          6132  4                        TOKEN[TOKEN$A_OPNAME], .TOKENLEN, TOKENBUFFER[1], 0)
;  6022          6133  4              THEN
;  6023          6134  5                  BEGIN
;  6024          6135  6                  IF (.OPERAND_EXPECTED AND
;  6025          6136  5                         (.TOKEN[TOKEN$B_KIND] EQL TOKEN$K_PREFIX_OP)) OR
;  6026          6137  6                     ((NOT .OPERAND_EXPECTED) AND
;  6027          6138  6                         (.TOKEN[TOKEN$B_KIND] NEQ TOKEN$K_PREFIX_OP))
;  6028          6139  5                  THEN
;  6029          6140  5                      RETURN .TOKEN
;  6030          6141  5
;  6031          6142  5                  ELSE
;  6032          6143  5                      BEST_TOKEN_FOUND = .TOKEN;
;  6033          6144  5
;  6034          6145  4                  END;
;  6035          6146  4
;  6036          6147  3              END;
;  6037          6148  3
;  6038          6149  3
;  6039          6150  3          ! If we found an operator but it is not in a valid context (e.g. infix
;  6040          6151  3          ! operator when we expect a prefix operator), we return it anyway but
;  6041          6152  3          ! only if it cannot be an address expression operator (which might be
;  6042          6153  3          ! valid in the current context).
;  6043          6154  3          !
;  6044          6155  3          IF (.BEST_TOKEN_FOUND NEQ 0) AND
;  6045          6156  4             NOT (.ADDRESS_EXPRESSION AND
;  6046          6157  4                  .CHARTBL[.STARTPTR[0], CHRTBL$V_ADDRESS_OP])
;  6047          6158  3          THEN
;  6048          6159  3              RETURN .BEST_TOKEN_FOUND;
;  6049          6160  3
;  6050          6161  3
;  6051          6162  3          ! There is no such operator.  Reset CHARPTR to point to the start of
;  6052          6163  3          ! the token to give the code below a chance to make sense of it.
;  6053          6164  3          !
;  6054          6165  3          CHARPTR = .STARTPTR;
;  6055          6166  3
;  6056          6167  2          END;                                ! End of operator scanning
;  6057          6168  2
;  6058          6169  2
;  6059          6170  2      ! If we are parsing an Address Expression at present, we check for the
;  6060          6171  2      ! specific operators allowed in Address Expressions.  We do not use the
;  6061          6172  2      ! language-specific rules for combining operator characters in this case.
;  6062          6173  2      ! If we find such an operator here, return the corresponding Token Entry.
```

```
;  6063          6174  2      ! If not, reset CHARPTR and continue through the Lexical Scanner.
;  6064          6175  2      !
;  6065          6176  2      IF .ADDRESS_EXPRESSION AND .CHARTBL[.CHARPTR[0], CHRTBL$V_ADDRESS_OP]
;  6066          6177  2      THEN
;  6067          6178  3          BEGIN
;  6068          6179  3
;  6069          6180  3          ! Special cases for C.
;  6070          6181  3          !
;  6071          6182  3          IF (.DBG$GB_LANGUAGE EQL DBG$K_C) AND
;  6072          6183  4             (.CHARPTR[0] EQL '-') AND (.CHARPTR[1] EQL '>')
;  6073          6184  3          THEN
;  6074          6185  4              BEGIN
;  6075          6186  4              CHARPTR = .CHARPTR + 2;
;  6076          6187  4              RETURN C_ARROW_TOKEN;
;  6077          6188  3              END;
;  6078          6189  3          IF (.DBG$GB_LANGUAGE EQL DBG$K_C) AND
;  6079          6190  3             (.CHARPTR[0] EQL '*') AND
;  6080          6191  4             (.OPERAND_EXPECTED)
;  6081          6192  3          THEN
;  6082          6193  4              BEGIN
;  6083          6194  4              CHARPTR = .CHARPTR + 1;
;  6084          6195  4              RETURN C_INDIRECT_TOKEN;
;  6085          6196  3              END;
;  6086          6197  3
;  6087          6198  3          ! Check for all other address expression operators.
;  6088          6199  3          !
;  6089          6200  3          CHARPTR = .CHARPTR + 1;
;  6090          6201  3          BEST_TOKEN_FOUND = 0;
;  6091          6202  3          INCR I FROM 0 TO .ADDR_EXPR_OPTBL[-1] - 1 DO
;  6092          6203  4              BEGIN
;  6093          6204  4              TOKEN = .ADDR_EXPR_OPTBL[.I] + TABLEBASE;
;  6094          6205  4              IF CHSEQL(.TOKEN[TOKEN$B_OPLEN],
;  6095          6206  4                        TOKEN[TOKEN$A_OPNAME], 1, STARTPTR[0], 0)
;  6096          6207  4              THEN
;  6097          6208  5                  BEGIN
;  6098          6209  6                  IF (.OPERAND_EXPECTED AND
;  6099          6210  5                          (.TOKEN[TOKEN$B_KIND] EQL TOKEN$K_PREFIX_OP)) OR
;  6100          6211  6                     ((NOT .OPERAND_EXPECTED) AND
;  6101          6212  6                          (.TOKEN[TOKEN$B_KIND] NEQ TOKEN$K_PREFIX_OP))
;  6102          6213  5                  THEN
;  6103          6214  5                      RETURN .TOKEN
;  6104          6215  5
;  6105          6216  5                  ELSE
;  6106          6217  5                      BEST_TOKEN_FOUND = .TOKEN;
;  6107          6218  5
;  6108          6219  4                  END;
;  6109          6220  4
;  6110          6221  3              END;
;  6111          6222  3
;  6112          6223  3          IF .BEST_TOKEN_FOUND NEQ 0 THEN RETURN .BEST_TOKEN_FOUND;
;  6113          6224  3          CHARPTR = .STARTPTR;
;  6114          6225  2          END;
;  6115          6226  2
;  6116          6227  2
;  6117          6228  2      ! So far we have not determined what kind of token we have here.  We thus
;  6118          6229  2      ! enter some language-specific code in a last-ditch effort to figure out
;  6119          6230  2      ! what kind of token we have.  If the language-specific code can recognize
```

```
; 6120          6231  2     ! a valid token, it returns a Lexical Token Entry for it.
; 6121          6232  2     !
; 6122          6233  2     CASE .DBG$GB_LANGUAGE FROM DBG$K_MIN_LANGUAGE TO DBG$K_MAX_LANGUAGE OF
; 6123          6234  2         SET
; 6124          6235  2
; 6125          6236  2
; 6126          6237  2         ! Handle Fortran.  Here we pick up the operators of the form .XX. or
; 6127          6238  2         ! .XXX. (such as .EQ., .NOT., etc.) and return the corresponding
; 6128          6239  2         ! Operator Token.  These operators are looked up in the FORTRAN
; 6129          6240  2         ! Special Operator Table.
; 6130          6241  2         !
; 6131          6242  2         [DBG$K_FORTRAN]:
; 6132          6243  3             BEGIN
; 6133          6244  3
; 6134          6245  3
; 6135          6246  3             ! Check for . as indirection operator in address expressions.
; 6136          6247  3             !
; 6137          6248  3             IF .ADDRESS_EXPRESSION AND .OPERAND_EXPECTED AND
; 6138          6249  4                (.CHARPTR[0] EQL '.')
; 6139          6250  3             THEN
; 6140          6251  4                 BEGIN
; 6141          6252  4                 CHARPTR = .CHARPTR + 1;
; 6142          6253  4                 RETURN FORTRAN_INDIRECT_TOKEN;
; 6143          6254  3                 END;
; 6144          6255  3
; 6145          6256  3
; 6146          6257  3             ! See if we have one of the FORTRAN relational or boolean
; 6147          6258  3             ! operators.
; 6148          6259  3             !
; 6149          6260  3             WHILE .CHARTBL[.CHARPTR[1], CHRTBL$V_ALPHABETIC] DO
; 6150          6261  3                 CHARPTR = .CHARPTR + 1;
; 6151          6262  3
; 6152          6263  3             CHARPTR = .CHARPTR + 2;
; 6153          6264  3             INCR I FROM 0 TO .FORTRAN_SPECIAL_OPTBL[-1] - 1 DO
; 6154          6265  4                 BEGIN
; 6155          6266  4                 TOKEN = .FORTRAN_SPECIAL_OPTBL[.I] + TABLEBASE;
; 6156          6267  4                 IF CH$EQL(.TOKEN[TOKEN$B_OPLEN], TOKEN[TOKEN$A_OPNAME],
; 6157          6268  4                                          .CHARPTR - .STARTPTR, .STARTPTR, 0)
; 6158          6269  4                 THEN
; 6159          6270  4                     RETURN .TOKEN;
; 6160          6271  4
; 6161          6272  3                 END;
; 6162          6273  3
; 6163          6274  3
; 6164          6275  3             ! Pick up FORTRAN Predefined Constant.
; 6165          6276  3             !
; 6166          6277  3             INCR I FROM 0 TO .PRIDTBL[-1] - 1 DO
; 6167          6278  4                 BEGIN
; 6168          6279  4                 PRID = .PRIDTBL[.I] + TABLEBASE;
; 6169          6280  4                 IF CH$EQL(.PRID[PRID$B_LENGTH], PRID[PRID$A_NAME],
; 6170          6281  4                                          .CHARPTR - .STARTPTR, .STARTPTR, 0)
; 6171          6282  4                 THEN
; 6172          6283  5                     BEGIN
; 6173          6284  5                     CH$MOVE(.PRID[PRID$B_LENGTH], PRID[PRID$A_NAME], TOKENBUFFER[1]);
; 6174          6285  5                     TOKENBUFFER[0] = .PRID[PRID$B_LENGTH];
; 6175          6286  5                     RETURN CREATE_OPERAND_TOKEN(TOKEN$K_IDENTIFIER, TOKENBUFFER);
; 6176          6287  4                     END;
```

```
;  6177     6288  4                END;
;  6178     6289  3
;  6179     6290  3
;  6180     6291  3
;  6181     6292  3            ! Return "." as the FORTRAN record component selection operator.
;  6182     6293  3            !
;  6183     6294  4            IF (NOT .OPERAND_EXPECTED) AND (.STARTPTR[0] EQL '.')
;  6184     6295  3            THEN
;  6185     6296  4                BEGIN
;  6186     6297  4                CHARPTR = .STARTPTR + 1;
;  6187     6298  4                RETURN FORTRAN_DOT_TOKEN;
;  6188     6299  3                END;
;  6189     6300  3
;  6190     6301  2            END;
;  6191     6302  2
;  6192     6303  2
;  6193     6304  2        ! Handle C.  Here we pick up the C operators prefix &, infix &,
;  6194     6305  2        ! infix &&, prefix and postfix ++, prefix and postfix --, +, -, and ->.
;  6195     6306  2        ! Possible ambiguities involving these operators must be
;  6196     6307  2        ! resolved using C rules, which is what we do here.
;  6197     6308  2        !
;  6198     6309  2        [DBG$K_C]:
;  6199     6310  3            BEGIN
;  6200     6311  3
;  6201     6312  3
;  6202     6313  3            ! Check for the operators that begin with '&': address-of,
;  6203     6314  3            ! bit-and, and short-and.
;  6204     6315  3            !
;  6205     6316  3            IF .CHARPTR[0] EQL '&'
;  6206     6317  3            THEN
;  6207     6318  4                BEGIN
;  6208     6319  4                CHARPTR = .CHARPTR + 1;
;  6209     6320  4                IF .CHARPTR[0] EQL '&'
;  6210     6321  4                THEN
;  6211     6322  5                    BEGIN
;  6212     6323  5                    CHARPTR = .CHARPTR + 1;
;  6213     6324  5                    RETURN C_AND_TOKEN;
;  6214     6325  4                    END;
;  6215     6326  4
;  6216     6327  4                IF .OPERAND_EXPECTED THEN RETURN C_ADDR_OF_TOKEN;
;  6217     6328  4                RETURN C_BIT_AND_TOKEN;
;  6218     6329  3                END;
;  6219     6330  3
;  6220     6331  3
;  6221     6332  3            ! Check for the operators that start with '+': add, pre-increment,
;  6222     6333  3            ! and post-increment. (X+Y  ++X  X++)
;  6223     6334  3            !
;  6224     6335  3            IF .CHARPTR[0] EQL '+'
;  6225     6336  3            THEN
;  6226     6337  4                BEGIN
;  6227     6338  4                CHARPTR = .CHARPTR + 1;
;  6228     6339  4                IF .CHARPTR[0] EQL '+'
;  6229     6340  4                THEN
;  6230     6341  5                    BEGIN
;  6231     6342  5                    CHARPTR = .CHARPTR + 1;
;  6232     6343  5                    IF .OPERAND_EXPECTED
;  6233     6344  5                    THEN
```

```
;  6234          6345  5
;  6235          6346  5
;  6236          6347  5                    ! Since ++ is not supported, put out an
;  6237          6348  5                    ! error message to that effect.
;  6238          6349  5                    ! If it does become supported then
;  6239          6350  5                    ! un-comment the commented out code
;  6240          6351  5                    ! and take out the error signal.
;  6241          6352  5                    !
;  6242          6353  5                    SIGNAL(DBG$_SIDEFFECT)
;  6243          6354  5                    ! RETURN C_PRE_INCR_TOKEN
;  6244          6355  5
;  6245          6356  5                ELSE
;  6246          6357  5                    SIGNAL(DBG$_SIDEFFECT);
;  6247          6358  5                    ! RETURN C_POST_INCR_TOKEN;
;  6248          6359  4                END;
;  6249          6360  4
;  6250          6361  4            RETURN C_ADD_TOKEN;
;  6251          6362  3            END;
;  6252          6363  3
;  6253          6364  3
;  6254          6365  3        ! Check for the operators that start with '-': subtract, pre-decrement,
;  6255          6366  3        ! and post-decrement, unary minus, and ->.
;  6256          6367  3        !
;  6257          6368  3        IF .CHARPTR[0] EQL '-'
;  6258          6369  3        THEN
;  6259          6370  4            BEGIN
;  6260          6371  4            CHARPTR = .CHARPTR + 1;
;  6261          6372  4            IF .CHARPTR[0] EQL '-'
;  6262          6373  4            THEN
;  6263          6374  5                BEGIN
;  6264          6375  5                CHARPTR = .CHARPTR + 1;
;  6265          6376  5                IF .OPERAND_EXPECTED
;  6266          6377  5                THEN
;  6267          6378  5
;  6268          6379  5
;  6269          6380  5                    ! Since -- is not supported, put out an
;  6270          6381  5                    ! error message to that effect.
;  6271          6382  5                    ! If it does become supported then
;  6272          6383  5                    ! un-comment the commented out code
;  6273          6384  5                    ! and take out the error signal.
;  6274          6385  5                    !
;  6275          6386  5                    SIGNAL(DBG$_SIDEFFECT)
;  6276          6387  5                    ! RETURN C_PRE_DECR_TOKEN
;  6277          6388  5                ELSE
;  6278          6389  5                    SIGNAL(DBG$_SIDEFFECT);
;  6279          6390  5                    ! RETURN C_POST_INCR_TOKEN;
;  6280          6391  4                END;
;  6281          6392  4
;  6282          6393  4            IF .CHARPTR[0] EQL '>'
;  6283          6394  4            THEN
;  6284          6395  5                BEGIN
;  6285          6396  5                CHARPTR = .CHARPTR + 1;
;  6286          6397  5                RETURN C_ARROW_TOKEN;
;  6287          6398  4                END;
;  6288          6399  4
;  6289          6400  4            IF .OPERAND_EXPECTED
;  6290          6401  4            THEN
```

```
;  6291     6402  4                    RETURN C_MINUS_TOKEN
;  6292     6403  4                ELSE
;  6293     6404  4                    RETURN C_SUB_TOKEN;
;  6294     6405  3                END;
;  6295     6406  3
;  6296     6407  2            END;
;  6297     6408  2
;  6298     6409  2
;  6299     6410  2        ! Handle RPG.  The special indicator names start with "*".
;  6300     6411  2        !
;  6301     6412  2        [DBG$K_RPG]:
;  6302     6413  3            BEGIN
;  6303     6414  3            IF .CHARPTR[0] EQL '*'
;  6304     6415  3            THEN
;  6305     6416  4                BEGIN
;  6306     6417  4
;  6307     6418  4                ! Pick it up as a name.
;  6308     6419  4                !
;  6309     6420  4                IF .OPERAND_EXPECTED
;  6310     6421  4                THEN
;  6311     6422  5                    BEGIN
;  6312     6423  5                    CHARPTR = .CHARPTR + 1;
;  6313     6424  5                    IF .CHARTBL[.CHARPTR[0], CHRTBL$V_IDENT_START]
;  6314     6425  5                    THEN
;  6315     6426  6                        BEGIN
;  6316     6427  6
;  6317     6428  6
;  6318     6429  6                        ! We have a valid start character for an identifier in the current
;  6319     6430  6                        ! language.  Now scan through the identifier as long as we have valid
;  6320     6431  6                        ! middle characters and set ENDPTR each time we find a valid end char-
;  6321     6432  6                        ! acter for an identifier.  At the end of the scan we set CHARPTR to
;  6322     6433  6                        ! point to the first character after the identifier end character.
;  6323     6434  6                        !
;  6324     6435  6                        ENDPTR = .CHARPTR - 2;
;  6325     6436  6                        WHILE TRUE DO
;  6326     6437  7                            BEGIN
;  6327     6438  7                            IF .CHARTBL[.CHARPTR[0], CHRTBL$V_IDENT_END]
;  6328     6439  7                            THEN
;  6329     6440  8                                BEGIN
;  6330     6441  8                                ENDPTR = .CHARPTR;
;  6331     6442  8                                IF NOT .CHARTBL[.CHARPTR[0], CHRTBL$V_IDENT_MIDDLE] THEN EXITLOOP;
;  6332     6443  7                                END;
;  6333     6444  7
;  6334     6445  7                            CHARPTR = .CHARPTR + 1;
;  6335     6446  7                            IF (NOT .CHARTBL[.CHARPTR[0], CHRTBL$V_IDENT_MIDDLE]) AND
;  6336     6447  8                               (NOT .CHARTBL[.CHARPTR[0], CHRTBL$V_IDENT_END])
;  6337     6448  7                            THEN EXITLOOP;
;  6338     6449  6                            END;
;  6339     6450  6
;  6340     6451  6                        CHARPTR = .ENDPTR + 1;
;  6341     6452  6
;  6342     6453  6
;  6343     6454  6                        ! Copy the identifier name to TOKENBUFFER.
;  6344     6455  6                        !
;  6345     6456  6                        TOKENLEN = .CHARPTR - .STARTPTR;
;  6346     6457  6                        IF .TOKENLEN GTR 255 THEN SIGNAL(DBG$_IDENTLONG);
;  6347     6458  6                        CH$MOVE(.TOKENLEN, .STARTPTR, TOKENBUFFER[1]);
```

```
;  6348          6459  6                        TOKENBUFFER[0] = .TOKENLEN;
;  6349          6460  6                        RETURN CREATE_OPERAND_TOKEN(TOKEN$K_IDENTIFIER, TOKENBUFFER);
;  6350          6461  5                        END;
;  6351          6462  5                    END
;  6352          6463  5
;  6353          6464  5                ! Pick it up as an operator.
;  6354          6465  5                !
;  6355          6466  4                ELSE
;  6356          6467  5                    BEGIN
;  6357          6468  5                    CHARPTR = .CHARPTR + 1;
;  6358          6469  5                    RETURN RPG_MULTIPLY_TOKEN;
;  6359          6470  4                    END;
;  6360          6471  4
;  6361          6472  3                END;
;  6362          6473  3
;  6363          6474  2            END;
;  6364          6475  2
;  6365          6476  2
;  6366          6477  2        ! Handle Ada.  Here we distinguish the tick operator from the single
;  6367          6478  2        ! quote--both are represented by the character "'".  If it is a tick,
;  6368          6479  2        ! we return the Tick Operator Token and if it is a single quote, we
;  6369          6480  2        ! pick up the character it quotes and return a Character Constant Token.
;  6370          6481  2        !
;  6371          6482  2        [DBG$K_ADA]:
;  6372          6483  3            BEGIN
;  6373          6484  3
;  6374          6485  3            IF .CHARPTR[0] EQL ''''
;  6375          6486  3            THEN
;  6376          6487  4                BEGIN
;  6377          6488  4
;  6378          6489  4
;  6379          6490  4                ! If we are expecting an infix or postfix operator, this must
;  6380          6491  4                ! be the "tick" character, which begins one of the postfix
;  6381          6492  4                ! tick operators (''FIRST", ''LAST", ...).
;  6382          6493  4                !
;  6383          6494  4                IF NOT .OPERAND_EXPECTED
;  6384          6495  4                THEN
;  6385          6496  5                    BEGIN
;  6386          6497  5
;  6387          6498  5                    ! Make a copy of the tick token.
;  6388          6499  5                    !
;  6389          6500  5                    TOKEN = DBG$GET_TEMPMEM (TOKEN$K_FIXED_SIZE_LONG + 4);
;  6390          6501  5                    CH$MOVE (TOKEN$K_FIXED_SIZE_BYTE+1,ADA_TICK_TOKEN,.TOKEN);
;  6391          6502  5
;  6392          6503  5                    ! Look up which tick operator this is.
;  6393          6504  5                    !
;  6394          6505  5                    INCR INDEX FROM TOKEN$K_TICK_MIN TO TOKEN$K_TICK_MAX DO
;  6395          6506  6                        BEGIN
;  6396          6507  6
;  6397          6508  6                        NAMEPTR = .ADA_TICK_TABLE[.INDEX] + TABLEBASE;
;  6398          6509  6
;  6399          6510  6                        IF CH$EQL (.NAMEPTR[0], NAMEPTR[1], .NAMEPTR[0], CHARPTR[1])
;  6400          6511  6                        THEN
;  6401          6512  7                            BEGIN
;  6402          6513  7                            CHARPTR = .CHARPTR + 1 + .NAMEPTR[0];
;  6403          6514  7                            TOKEN[TOKEN$W_SUBCODE] = .INDEX;
;  6404          6515  7                            TOKEN[TOKEN$B_OPLEN] = .NAMEPTR[0] + 1;
```

```
; 6405          6516  7                 CH$COPY(1, CH$PTR(UPLIT('''')), .NAMEPTR[0], NAMEPTR[1],
; 6406          6517  7                         %C' ', .NAMEPTR[0]+1, TOKEN[TOKEN$A_OPNAME]);
; 6407          6518  7
; 6408          6519  7                 ! Look for a left paren that would indicate a
; 6409          6520  7                 ! list of arguments follow the tick operator.
; 6410          6521  7                 ! If a paren is found, the subcode value is set
; 6411          6522  7                 ! set to the corresponding tick operator type by
; 6412          6523  7                 ! adding 1 to it.  Also, the CHARPTR will be
; 6413          6524  7                 ! updated to pointer at the character just past
; 6414          6525  7                 ! the paren.
; 6415          6526  7                 !
; 6416          6527  7                 WHILE .CHARTBL[.CHARPTR[0], CHRTBL$V_SPACE] DO
; 6417          6528  7                     CHARPTR = .CHARPTR + 1;
; 6418          6529  7
; 6419          6530  7                 IF .CHARPTR[0] EQL '('
; 6420          6531  7                 THEN
; 6421          6532  8                     BEGIN
; 6422          6533  8                     TOKEN[TOKEN$V_ARGUMENT_LIST] = TRUE;
; 6423          6534  8                     CHARPTR = .CHARPTR + 1;
; 6424          6535  8                     END
; 6425          6536  7                 ELSE
; 6426          6537  7                     TOKEN[TOKEN$V_ARGUMENT_LIST] = FALSE;
; 6427          6538  7
; 6428          6539  7                 RETURN .TOKEN;
; 6429          6540  6                 END;
; 6430          6541  5             END;
; 6431          6542  5
; 6432          6543  5         ! If we fall through to here, we failed to find
; 6433          6544  5         ! a matching tick operator in our table.
; 6434          6545  5         !
; 6435          6546  5         TOKENBUFFER[0] = 1;
; 6436          6547  5         TOKENBUFFER[1] = '''';
; 6437          6548  5         CHARPTR = .CHARPTR + 1;
; 6438          6549  5         WHILE .CHARTBL[.CHARPTR[0], CHRTBL$V_IDENT_START] OR
; 6439          6550  5               .CHARTBL[.CHARPTR[0], CHRTBL$V_IDENT_MIDDLE] OR
; 6440          6551  5               .CHARTBL[.CHARPTR[0], CHRTBL$V_IDENT_END] DO
; 6441          6552  6             BEGIN
; 6442          6553  7             IF (.CHARPTR[0] EQL CAR_RET) OR (.TOKENBUFFER[0] GEQ 32)
; 6443          6554  6                 THEN EXITLOOP;
; 6444          6555  6             TOKENBUFFER[0] = .TOKENBUFFER[0] + 1;
; 6445          6556  6             TOKENBUFFER[.TOKENBUFFER[0]] = .CHARPTR[0];
; 6446          6557  6             CHARPTR = .CHARPTR + 1;
; 6447          6558  5             END;
; 6448          6559  5
; 6449          6560  5         SIGNAL(DBG$_UNKATTRIB, 1, TOKENBUFFER);
; 6450          6561  4         END;
; 6451          6562  4
; 6452          6563  4         ! Otherwise we are expecting an operand, so it must be the
; 6453          6564  4         ! single quote character.  Pick up the single character quoted
; 6454          6565  4         ! and return a Character Constant Lexical Token Entry.
; 6455          6566  4         !
; 6456          6567  5         IF (.CHARPTR[1] EQL CAR_RET) OR (.CHARPTR[2] NEQ '''')
; 6457          6568  4         THEN
; 6458          6569  4             SIGNAL(DBG$_INVCHRCON);
; 6459          6570  4
; 6460          6571  4         TOKENBUFFER[0] = 3;
; 6461          6572  4         CH$MOVE(3, CHARPTR[0], TOKENBUFFER[1]);
```

```
; 6462      6573  4                 CHARPTR = .CHARPTR + 3;
; 6463      6574  4                 RETURN CREATE_OPERAND_TOKEN(TOKEN$K_IDENTIFIER, TOKENBUFFER);
; 6464      6575  3                 END;
; 6465      6576  3
; 6466      6577  2             END;
; 6467      6578  2
; 6468      6579  2
; 6469      6580  2         ! Do nothing for all other languages.
; 6470      6581  2         !
; 6471      6582  2         [INRANGE, OUTRANGE]:
; 6472      6583  2             0;
; 6473      6584  2
; 6474      6585  2         TES;
; 6475      6586  2
; 6476      6587  2
; 6477      6588  2     ! We have not found a valid token yet.  This must therefore be a genuine
; 6478      6589  2     ! syntax error, so we signal an appropriate error message.
; 6479      6590  2     !
; 6480      6591  2     CHARPTR = .STARTPTR;
; 6481      6592  2     TOKENBUFFER[0] = 0;
; 6482      6593  2     INCR I FROM 0 TO 20 DO
; 6483      6594  3         BEGIN
; 6484      6595  3         IF .CHARPTR[.I] EQL CAR_RET THEN EXITLOOP;
; 6485      6596  3         TOKENBUFFER[.I + 1] = .CHARPTR[.I];
; 6486      6597  3         TOKENBUFFER[0] = .TOKENBUFFER[0] + 1;
; 6487      6598  2         END;
; 6488      6599  2
; 6489      6600  2     SIGNAL(DBG$_SYNERREXPR, 1, TOKENBUFFER);
; 6490      6601  2     RETURN 0;
; 6491      6602  2
; 6492      6603  1     END;
; INFO#250                    L1:5832
; Referenced LOCAL symbol BACKUP_DIGIT_PTR is probably not initialized


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

49 58 45 4C 5C 52 45 53 52 41 50 47 42 44 1C 03117 P.AWZ:  .ASCII  <28>\DBGPARSER\<92>\LEXICAL_SCANNER 10\      ;
   30 31 20 52 45 4E 4E 41 43 53 5F 4C 41 43 03126                                                            ;
                           20 45 4E 49 4C 25 03134 P.AXA:  .ASCII  \%LINE \                                   ;
                        20 4C 45 42 41 4C 25 0313A P.AXB:  .ASCII  \%LABEL \                                  ;
49 58 45 4C 5C 52 45 53 52 41 50 47 42 44 1C 03141 P.AXC:  .ASCII  <28>\DBGPARSER\<92>\LEXICAL_SCANNER 20\      ;
   30 32 20 52 45 4E 4E 41 43 53 5F 4C 41 43 03150                                                            ;
49 58 45 4C 5C 52 45 53 52 41 50 47 42 44 1C 0315E P.AXD:  .ASCII  <28>\DBGPARSER\<92>\LEXICAL_SCANNER 30\      ;
   30 33 20 52 45 4E 4E 41 43 53 5F 4C 41 43 0316D                                                            ;
24 47 42 44 5C 52 45 53 52 41 50 47 42 44 20 0317B P.AXE:  .ASCII  \ DBGPARSER\<92>\DBG$LEXICAL_SCANNER 40\     ;
52 45 4E 4E 41 43 53 5F 4C 41 43 49 58 45 4C 0318A                                                            ;
                                    30 34 20 03199                                                            ;
                                 00 00 00 27 0319C P.AXF:  .ASCII  \'\<0><0><0>                                ;


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

                                                 0FFC 00000         .ENTRY  DBG$LEXICAL_SCANNER, Save R2,R3,R4,R5,R6,-  ; 5034
                                                                            R7,R8,R9,R10,R11                          ;
```

```
                                5E      FEEC    CE    9E 00002          MOVAB   -276(SP), SP
                                50 00000000'    FF    9A 00007 1$:      MOVZBL  @CHARPTR, R0                                    ;  5149
                                   00000000'EF40    DF 0000E           PUSHAL  CHARTBL+2[R0]
                        08          9E           01    E1 00015          BBC     #1, @(SP)+, 2$
                                   00000000'     EF    D6 00019          INCL    CHARPTR                                         ;  5150
                                                E6    11 0001F          BRB     1$
                                54 00000000'    EF    D0 00021 2$:      MOVL    CHARPTR, R4                                     ;  5152
                                56              54    D0 00028          MOVL    R4, STARTPTR
                                57              64    9A 0002B          MOVZBL  (R4), R7                                        ;  5157
                                0D              57    91 0002E          CMPB    R7, #13
                                                08    12 00031          BNEQ    3$
                                   00000000'    EF    7C 00033          CLRQ    TERMINATOR_CODE                                 ;  5160
                                                6C    11 00039          BRB     8$                                              ;  5162
                                5B 00000000'EF47    DE 0003B 3$:      MOVAL   CHARTBL[R7], R11                                ;  5171
                        6D      6B              12    E1 00043          BBC     #18, (R11), 10$
                                59          0C  AC    D0 00047          MOVL    TERM_LIST, R9                                   ;  5178
                                58              01    CE 0004B          MNEGL   #1, I
                                                5F    11 0004E          BRB     9$
                                50 00000000'    EF    9E 00050 4$:      MOVAB   TABLEBASE, R0                                   ;  5180
                        55      50            6948    C1 00057          ADDL3   (R9)[I], R0, TERMPTR
                                5A          08  A5    9A 0005C          MOVZBL  8(TERMPTR), R10                                 ;  5188
                        64  09  A5              5A    29 00060          CMPC3   R10, 9(TERMPTR), (R4)
                                                48    12 00065          BNEQ    9$
                                1A          01  AB    E9 00067          BLBC    1(R11), 5$                                      ;  5193
                                50            6A44    9A 0006B          MOVZBL  (R10)[R4], R0                                   ;  5195
                                   00000000'EF40    DF 0006F           PUSHAL  CHARTBL[R0]
                        35          9E           01    E0 00076          BBS     #1, @(SP)+, 9$
                                   00000000'EF40    DF 0007A           PUSHAL  CHARTBL[R0]                                     ;  5197
                        2A          9E           02    E0 00081          BBS     #2, @(SP)+, 9$
                                    05          01  A5    E9 00085 5$:      BLBC    1(TERMPTR), 6$                          ;  5201
                                                10  AC    D5 00089          TSTL    PAREN_NESTING
                                                21    12 0008C          BNEQ    9$
                        06                      65    09    E1 0008E 6$:      BBC     #9, (TERMPTR), 7$                       ;  5205
                                                57    01    A4 91 00092       CMPB    1(R4), R7                               ;  5206
                                                      17    13 00096          BEQL    9$
                                00000000'       EF    02    A5 3C 00098 7$:   MOVZWL  2(TERMPTR), TERMINATOR_CODE             ;  5215
                                00000000'       EF          5A D0 000A0       MOVL    R10, TERMINATOR_LENGTH                  ;  5216
                                                50 00000000'  EF 9E 000A7 8$:   MOVAB   TERMINATOR_TOKEN, R0                    ;  5217
                                                              04 000AE          RET
                        9C                      58    FC    A9 F2 000AF 9$:   AOBLSS  -4(R9), I, 4$                          ;  5178
                        4F                      6B          13 E1 000B4 10$:    BBC     #19, (R11), 14$                       ;  5231
                        0A                      00 00000000G  00 8F 000B8       CASEB   DBG$GB_LANGUAGE, #0, #10              ;  5234
0016                    0016                  0016          0016    000C0 11$:    .WORD   12$-11$,-
0016                    0016                  002D          0016    000C8         12$-11$,-
                        0016                  0016          0016    000D0         12$-11$,-
                                                                                  12$-11$,-
                                                                                  12$-11$,-
                                                                                  13$-11$,-
                                                                                  12$-11$,-
                                                                                  12$-11$,-
                                                                                  12$-11$,-
                                                                                  12$-11$,-
                                                                                  12$-11$
                                                00000000'     EF    9F 000D6 12$:     PUSHAB  P.AWZ                             ;  5260
                                                              01    DD 000DC          PUSHL   #1
                                                00028362      8F    DD 000DE          PUSHL   #164706
                                00000000G       00                  03 FB 000E4       CALLS   #3, LIB$SIGNAL
```

```
                             1A  11 000EB          BRB      14$                                  ;
         2D                  57  91 000ED 13$:     CMPB     R7, #45                              ; 5246
                             15  12 000F0          BNEQ     14$                                  ;
         3E          01      A4  91 000F2          CMPB     1(R4), #^2                           ;
                             0F  12 000F6          BNEQ     14$                                  ;
00000000'  EF                02  C0 000F8          ADDL2    #2, CHARPTR                          ; 5249
         50 00000000'        EF  9E 000FF          MOVAB    PLI_ARROW_TOKEN, R0                  ; 5250
                                 04 00106          RET                                           ;
     0C  AE          04      AC  D0 00107 14$:     MOVL     OPERAND_EXPECTED, 12(SP)             ; 5273
         03          0C      AE  E8 0010C          BLBS     12(SP), 16$                          ;
                           00FB  31 00110 15$:     BRW      28$                                  ;
         50 00000000'        FF  9A 00113 16$:     MOVZBL   @CHARPTR, R0                         ;
         EE 00000000'EF40        E9 0011A          BLBC     CHARTBL+2[R0], 15$                   ;
            00000000'        EF  D6 00122 17$:     INCL     CHARPTR                              ; 5276
         50 00000000'        FF  9A 00128          MOVZBL   @CHARPTR, R0                         ; 5277
            00000000'EF40        DF 0012F          PUSHAL   CHARTBL+2[R0]                        ;
E8          9E               01  E0 00136          BBS      #1, @(SP)+, 17$                      ;
         50 00000000G        00  E9 0013A          BLBC     DBG$GB_SET_BREAK_FLAG, 20$           ; 5285
         50 00000000'        EF  D0 00141          MOVL     CHARPTR, R0                          ; 5288
     44  8F                  60  91 00148          CMPB     (R0), #68                            ;
                             1C  12 0014C          BNEQ     19$                                  ;
     4F  8F          01      A0  91 0014E          CMPB     1(R0), #79                           ; 5289
                             15  12 00153          BNEQ     19$                                  ;
         20          02      A0  91 00155          CMPB     2(R0), #32                           ; 5290
                             06  13 00159          BEQL     18$                                  ;
         28          02      A0  91 0015B          CMPB     2(R0), #40                           ;
                             09  12 0015F          BNEQ     19$                                  ;
         51 00000000'        EF  9E 00161 18$:     MOVAB    CURLOC_TOKEN, R1                     ; 5292
                             7C  11 00168          BRB      24$                                  ;
     57  8F                  60  91 0016A 19$:     CMPB     (R0), #87                            ; 5293
                             21  12 0016E          BNEQ     20$                                  ;
     48  8F          01      A0  91 00170          CMPB     1(R0), #72                           ; 5294
                             1A  12 00175          BNEQ     20$                                  ;
     45  8F          02      A0  91 00177          CMPB     2(R0), #69                           ; 5295
                             13  12 0017C          BNEQ     20$                                  ;
     4E  8F          03      A0  91 0017E          CMPB     3(R0), #78                           ; 5296
                             0C  12 00183          BNEQ     20$                                  ;
         20          04      A0  91 00185          CMPB     4(R0), #32                           ; 5297
                             74  13 00189          BEQL     26$                                  ;
         28          04      A0  91 0018B          CMPB     4(R0), #40                           ;
                             6E  13 0018F          BEQL     26$                                  ;
         50 00000000'        FF  9A 00191 20$:     MOVZBL   @CHARPTR, R0                         ; 5331
         67 00000000'EF40        E8 00198          BLBS     CHARTBL[R0], 27$                     ;
            00000000'EF40        DF 001A0          PUSHAL   CHARTBL+1[R0]                        ; 5332
5C          9E               01  E0 001A7          BBS      #1, @(SP)+, 27$                      ;
            25               50  91 001AB          CMPB     R0, #37                              ; 5333
                             0D  12 001AE          BNEQ     21$                                  ;
         07 00000000G        00  91 001B0          CMPB     DBG$GB_LANGUAGE, #7                  ; 5334
                             4E  12 001B7          BNEQ     27$                                  ;
         4A          08      AC  E8 001B9          BLBS     ADDRESS_EXPRESSION, 27$              ;
     5C  8F                  66  91 001BD 21$:     CMPB     (STARTPTR), #92                      ; 5337
                             09  12 001C1          BNEQ     22$                                  ;
         51 00000000'        EF  9E 001C3          MOVAB    CURVAL_TOKEN, R1                     ;
                             1A  11 001CA          BRB      24$                                  ;
     5E  8F                  66  91 001CC 22$:     CMPB     (STARTPTR), #94                      ; 5338
                             18  12 001D0          BNEQ     25$                                  ;
         05 00000000G        00  91 001D2          CMPB     DBG$GB_LANGUAGE, #5                  ; 5339
```

```
                                     04   12 001D9        BNEQ    23$                                         ;
                     0B         08   AC   E9 001DB        BLBC    ADDRESS_EXPRESSION, 25$                     ;
                     51 00000000'    EF   9E 001DF 23$:   MOVAB   PREVLOC_TOKEN, R1                           ; 5341
                     50              51   D0 001E6 24$:   MOVL    R1, R0                                      ;
                                     04      001E9        RET                                                 ;
                     2E              66   91 001EA 25$:   CMPB    (STARTPTR), #46                             ; 5342
                                     18   12 001ED        BNEQ    27$                                         ;
                     28              50   91 001EF        CMPB    R0, #40                                     ; 5343
                                     13   13 001F2        BEQL    27$                                         ;
                     2E              50   91 001F4        CMPB    R0, #46                                     ; 5344
                                     0E   13 001F7        BEQL    27$                                         ;
                5C   8F              50   91 001F9        CMPB    R0, #92                                     ; 5345
                                     08   13 001FD        BEQL    27$                                         ;
                     50 00000000'    EF   9E 001FF 26$:   MOVAB   CURLOC_TOKEN, R0                            ; 5347
                                     04      00206        RET                                                 ;
           00000000' EF              56   D0 00207 27$:   MOVL    STARTPTR, CHARPTR                           ; 5351
                     50 00000000'    EF   D0 0020E 28$:   MOVL    CHARPTR, R0                                 ; 5359
                     25              60   91 00215        CMPB    (R0), #37                                   ;
                                     03   13 00218        BEQL    30$                                         ;
                                   02C8   31 0021A 29$:   BRW     68$                                         ;
                     F9         0C   AE   E9 0021D 30$:   BLBC    12(SP), 29$                                 ; 5360
                     50         01   A0   9A 00221        MOVZBL  1(R0), R0                                   ; 5361
                     ED 00000000'EF40     E9 00225        BLBC    CHARTBL+1[R0], 29$                          ;
                        00000000'    EF   D6 0022D        INCL    CHARPTR                                     ; 5368
                     59              01   D0 00233        MOVL    #1, TOKENLEN                                ; 5369
                15   AE              25   90 00236        MOVB    #37, TOKENBUFFER+1                          ; 5370
                     50 00000000'    FF   9A 0023A 31$:   MOVZBL  @CHARPTR, R0                                ; 5371
                     0B 00000000'EF40     E8 00241        BLBS    CHARTBL+1[R0], 32$                          ;
                        00000000'EF40     DF 00249        PUSHAL  CHARTBL+1[R0]                               ; 5372
           0F        9E              01   E1 00250        BBC     #1, @(SP)+, 33$                             ;
                                     59   D6 00254 32$:   INCL    TOKENLEN                                    ; 5375
                14 AE49              50   90 00256        MOVB    R0, TOKENBUFFER[TOKENLEN]                   ; 5376
                        00000000'    EF   D6 0025B        INCL    CHARPTR                                     ; 5377
                                     D7   11 00261        BRB     31$                                         ; 5371
                14   AE              59   90 00263 33$:   MOVB    TOKENLEN, TOKENBUFFER                       ; 5380
                                     55   D4 00267        CLRL    INDEX                                       ; 5385
                     54              01   CE 00269        MNEGL   #1, I                                       ; 5386
                                     23   11 0026C        BRB     35$                                         ;
                     50 00000000'    EF   9E 0026E 34$:   MOVAB   TABLEBASE, R0                               ; 5388
           5B        50 00000000'EF44     C1 00275        ADDL3   PERCENT_TABLE[I], R0, NAMEPTR               ;
                     50         01   AB   9A 0027E        MOVZBL  1(NAMEPTR), R0                              ; 5389
   59      00   02   AB              50   2D 00282        CMPC5   R0, 2(NAMEPTR), #0, TOKENLEN, TOKENBUFFER+1 ;
                                15   AE      00288                                                        ;
                                     05   12 0028A        BNEQ    35$                                         ;
                     55              6B   9A 0028C        MOVZBL  (NAMEPTR), INDEX                            ; 5392
                                     08   11 0028F        BRB     36$                                         ; 5391
           D5        54 00000000'    EF   F2 00291 35$:   AOBLSS  PERCENT_TABLE-4, I, 34$                     ; 5386
           08        00              55   CF 00299 36$:   CASEL   INDEX, #0, #8                               ; 5401
 019C      002A      002A          0241      0029D 37$:   .WORD   67$-37$,-                                   ;
 0239      0231      0229          0221      002A5                38$-37$,-                                   ;
                                   0B39      002AD                38$-37$,-                                   ;
                                                                  54$-37$,-                                   ;
                                                                  63$-37$,-                                   ;
                                                                  64$-37$,-                                   ;
                                                                  65$-37$,-                                   ;
                                                                  66$-37$,-                                   ;
                                                                  201$-37$                                    ;
```

```
                                00000000'  EF   9F 002AF            PUSHAB  P.AXC                                   ; 5611
                                           01   DD 002B5            PUSHL   #1                                      ;
                                00028362   8F   DD 002B7            PUSHL   #164706                                 ;
                    00000000G   00         03   FB 002BD            CALLS   #3, LIB$SIGNAL                          ;
                                         0217   31 002C4            BRW     67$                                     ;
                                           57   D4 002C7 38$:       CLRL    R7                                      ; 5424
                                01         55   D1 002C9            CMPL    INDEX, #1                               ;
                                           17   12 002CC            BNEQ    39$                                     ;
                                           57   D6 002CE            INCL    R7                                      ;
                15  AE 00000000'   EF      06   28 002D0            MOVC3   #6, P.AXA, TOKENBUFFER+1                ; 5427
                                   59      06   D0 002D9            MOVL    #6, TOKENLEN                            ; 5428
                                   52 000289F2   8F D0 002DC        MOVL    #166386, ERRORMSG                       ; 5429
                                           13   11 002E3            BRB     40$                                     ; 5424
                15  AE 00000000'   EF      07   28 002E5 39$:       MOVC3   #7, P.AXB, TOKENBUFFER+1                ; 5434
                                   59      07   D0 002EE            MOVL    #7, TOKENLEN                            ; 5435
                                   52 000289EA   8F D0 002F1        MOVL    #166378, ERRORMSG                       ; 5436
                                   50 00000000'  FF 9A 002F8 40$:   MOVZBL  @CHARPTR, R0                            ; 5443
                                00000000'EF40    DF 002FF            PUSHAL  CHARTBL+2[R0]                           ;
                        09         9E           01   E0 00306       BBS     #1, @(SP)+, 41$                         ;
                                           52   DD 0030A            PUSHL   ERRORMSG                                ;
                    00000000G   00         01   FB 0030C            CALLS   #1, LIB$SIGNAL                          ;
                                   50 00000000'  FF 9A 00313 41$:   MOVZBL  @CHARPTR, R0                            ; 5444
                                00000000'EF40    DF 0031A            PUSHAL  CHARTBL+2[R0]                           ;
                        08         9E           01   E1 00321       BBC     #1, @(SP)+, 42$                         ;
                                00000000'  EF      D6 00325            INCL    CHARPTR                             ; 5445
                                           E6   11 0032B            BRB     41$                                     ;
                                   50 00000000'  FF 9A 0032D 42$:   MOVZBL  @CHARPTR, R0                            ; 5450
                                00000000'EF40    DF 00334            PUSHAL  CHARTBL+1[R0]                           ;
            09                     9E           01   E0 0033B       BBS     #1, @(SP)+, 43$                         ;
                                           52   DD 0033F            PUSHL   ERRORMSG                                ; 5452
                    00000000G   00         01   FB 00341            CALLS   #1, LIB$SIGNAL                          ;
                                   50 00000000'  EF D0 00348 43$:   MOVL    CHARPTR, R0                             ; 5454
                                   30          60   91 0034F        CMPB    (R0), #48                               ;
                                           17   12 00352            BNEQ    44$                                     ;
                                   50      01   A0 9A 00354        MOVZBL  1(R0), R0                               ; 5455
                                00000000'EF40    DF 00358            PUSHAL  CHARTBL+1[R0]                           ;
            08                     9E           01   E1 0035F       BBC     #1, @(SP)+, 44$                         ;
                                00000000'  EF      D6 00363            INCL    CHARPTR                             ; 5457
                                           DD   11 00369            BRB     43$                                     ;
                                   50 00000000'  FF 9A 0036B 44$:   MOVZBL  @CHARPTR, R0                            ; 5463
                                00000000'EF40    DF 00372            PUSHAL  CHARTBL+1[R0]                           ;
            25                     9E           01   E1 00379       BBC     #1, @(SP)+, 46$                         ;
                    000000FF    8F             59   D1 0037D        CMPL    TOKENLEN, #255                          ; 5465
                                           09   19 00384            BLSS    45$                                     ;
                                           52   DD 00386            PUSHL   ERRORMSG                                ;
                    00000000G   00         01   FB 00388            CALLS   #1, LIB$SIGNAL                          ;
                                           59   D6 0038F 45$:       INCL    TOKENLEN                                ; 5466
                          14 AE49 00000000'  FF 90 00391            MOVB    @CHARPTR, TOKENBUFFER[TOKENLEN]         ; 5467
                                00000000'  EF      D6 0039A            INCL    CHARPTR                             ; 5468
                                           C9   11 003A0            BRB     44$                                     ; 5463
                                   03          57   E8 003A2 46$:   BLBS    R7, 48$                                 ; 5476
                                         010F   31 003A5 47$:       BRW     61$                                     ;
                                   2E 00000000'  FF 91 003A8 48$:   CMPB    @CHARPTR, #46                           ;
                                           F4   12 003AF            BNEQ    47$                                     ;
                                   50 00000000'  EF D0 003B1        MOVL    CHARPTR, R0                             ; 5483
                                           50   D6 003B8            INCL    R0                                      ;
                                   50          60   9A 003BA        MOVZBL  (R0), R0                                ;
```

```
                    00000000'EF40   DF 003BD          PUSHAL  CHARTBL+1[R0]                                   ;
09                        9E        01 E1 003C4          BBC     #1, @(SP)+, 49$                                 ;
   000000FF               8F        59 D1 003C8          CMPL    TOKENLEN, #255                                  ; 5484
                                    09 19 003CF          BLSS    50$                                             ;
                                    52 DD 003D1 49$:     PUSHL   ERRORMSG                                        ; 5486
   00000000G               00       01 FB 003D3          CALLS   #1, LIB$SIGNAL                                  ;
                                    59 D6 003DA 50$:     INCL    TOKENLEN                                        ; 5488
         14 AE49                    2E 90 003DC          MOVB    #46, TOKENBUFFER[TOKENLEN]                      ; 5489
                    00000000'       EF D6 003E1 51$:     INCL    CHARPTR                                         ; 5490
                         50 00000000' EF D0 003E7          MOVL    CHARPTR, R0                                     ; 5495
                         30         60 91 003EE          CMPB    (R0), #48                                       ;
                                    0F 12 003F1          BNEQ    52$                                             ;
                         50       01 A0 9A 003F3          MOVZBL  1(R0), R0                                       ; 5496
                    00000000'EF40   DF 003F7          PUSHAL  CHARTBL+1[R0]                                   ;
DF                        9E        01 E0 003FE          BBS     #1, @(SP)+, 51$                                 ;
                         50 00000000' FF 9A 00402 52$:     MOVZBL  @CHARPTR, R0                                    ; 5503
                    00000000'EF40   DF 00409          PUSHAL  CHARTBL+1[R0]                                   ;
91                        9E        01 E1 00410          BBC     #1, @(SP)+, 47$                                 ;
   000000FF               8F        59 D1 00414          CMPL    TOKENLEN, #255                                  ; 5505
                                    09 19 0041B          BLSS    53$                                             ;
                                    52 DD 0041D          PUSHL   ERRORMSG                                        ;
   00000000G               00       01 FB 0041F          CALLS   #1, LIB$SIGNAL                                  ;
                                    59 D6 00426 53$:     INCL    TOKENLEN                                        ; 5506
         14 AE49                    FF 90 00428          MOVB    @CHARPTR, TOKENBUFFER[TOKENLEN]                 ; 5507
                    00000000'       EF D6 00431          INCL    CHARPTR                                         ; 5508
                                    C9 11 00437          BRB     52$                                             ; 5503
                         50 00000000' FF 9A 00439 54$:     MOVZBL  @CHARPTR, R0                                    ; 5533
                    00000000'EF40   DF 00440          PUSHAL  CHARTBL+2[R0]                                   ;
08                        9E        01 E1 00447          BBC     #1, @(SP)+, 55$                                 ;
                    00000000'       EF D6 0044B          INCL    CHARPTR                                         ; 5534
                                    E6 11 00451          BRB     54$                                             ;
                         56 00000000' EF D0 00453 55$:     MOVL    CHARPTR, STARTPTR                               ; 5536
                         50 00000000' FF 9A 0045A          MOVZBL  @CHARPTR, R0                                    ; 5542
                    00000000'EF40   DF 00461          PUSHAL  CHARTBL+1[R0]                                   ;
      0D                  9E        02 E1 00468          BBC     #2, @(SP)+, 56$                                 ;
                              10    AE 9F 0046C          PUSHAB  TOKEN_TYPE                                      ; 5544
                              18    AE 9F 0046F          PUSHAB  TOKENBUFFER                                     ;
            0000V             CF      02 FB 00472          CALLS   #2, SCAN_QUOTED_STRING                          ;
                                    42 11 00477          BRB     62$                                             ;
                         50 00000000' FF 9A 00479 56$:     MOVZBL  @CHARPTR, R0                                    ; 5553
                         50 00000000'EF40 DE 00480          MOVAL   CHARTBL[R0], R0                                 ;
                         08         60 E8 00488          BLBS    (R0), 57$                                       ;
      04                 60         01 E0 0048B          BBS     #1, (R0), 57$                                   ; 5554
      08                 60         02 E1 0048F          BBC     #2, (R0), 58$                                   ; 5555
                    00000000'       EF D6 00493 57$:     INCL    CHARPTR                                         ; 5557
                                    DE 11 00499          BRB     56$                                             ;
      59 00000000'      EF          56 C3 0049B 58$:     SUBL3   STARTPTR, CHARPTR, TOKENLEN                     ; 5559
                                    0D 12 004A3          BNEQ    60$                                             ; 5560
                    000289BA        8F DD 004A5          PUSHL   #166330                                         ;
   00000000G                00      01 FB 004AB 59$:     CALLS   #1, LIB$SIGNAL                                  ;
15 AE                       66      59 28 004B2 60$:     MOVC3   TOKENLEN, (STARTPTR), TOKENBUFFER+1             ; 5561
         14 AE                      59 90 004B7 61$:     MOVB    TOKENLEN, TOKENBUFFER                           ; 5562
                              0918  31 004BB 62$:     BRW     201$                                            ; 5577
                         50 00000000' EF 9E 004BE 63$:     MOVAB   RADIX_OP_DEC, R0                                ; 5584
                                    04 004C5          RET                                                     ;
                         50 00000000' EF 9E 004C6 64$:     MOVAB   RADIX_OP_HEX, R0                                ; 5591
                                    04 004CD          RET                                                     ;
```

```
                50 00000000'   EF   9E 004CE 65$:     MOVAB   RADIX_OP_OCT, R0                        ; 5598
                               04   004D5              RET
                50 00000000'   EF   9E 004D6 66$:     MOVAB   RADIX_OP_BIN, R0                        ; 5605
                               04   004DD              RET
    00000000'   EF             56   D0 004DE 67$:     MOVL    STARTPTR, CHARPTR                       ; 5621
                50 00000000'   FF   9A 004E5 68$:     MOVZBL  @CHARPTR, R0                            ; 5634
                   00000000'EF40   DF 004EC           PUSHAL  CHARTBL+1[R0]
14              9E             02   E1 004F3           BBC     #2, @(SP)+, 69$
                         10    AE   9F 004F7           PUSHAB  TOKEN_TYPE                              ; 5637
                         18    AE   9F 004FA           PUSHAB  TOKENBUFFER
          0000V CF             02   FB 004FD           CALLS   #2, SCAN_QUOTED_STRING
                         14    AE   9F 00502           PUSHAB  TOKENBUFFER                             ; 5638
                         14    AE   DD 00505           PUSHL   TOKEN_TYPE
                            08D0   31 00508           BRW     202$
                50 00000000'   FF   9A 0050B 69$:     MOVZBL  @CHARPTR, R0                            ; 5655
                   00000000'EF40   DF 00512           PUSHAL  CHARTBL[R0]
03              9E             03   E0 00519           BBS     #3, @(SP)+, 71$
                            01D8   31 0051D 70$:     BRW     107$
                05        08   AC   E9 00520 71$:     BLBC    ADDRESS_EXPRESSION, 72$                 ; 5656
                2E             50   91 00524           CMPB    R0, #46
                               F4   13 00527           BEQL    70$
                               58   D4 00529 72$:     CLRL    STATE_INDEX                             ; 5669
                09 00000000G   00   91 0052B           CMPB    DBG$GB_LANGUAGE, #9                     ; 5670
                               0C   12 00532           BNEQ    73$
                0A 00000000'   EF   D1 00534           CMPL    EXPRESSION_RADIX, #10                   ; 5671
                               03   13 0053B           BEQL    73$
                58             1F   D0 0053D           MOVL    #31, STATE_INDEX                        ; 5673
                57             04   D0 00540 73$:     MOVL    #4, NUMBER_KIND                         ; 5675
                50 00000000'   EF   D0 00543           MOVL    EXPRESSION_RADIX, R0                    ; 5676
                10             50   D1 0054A           CMPL    R0, #16
                               03   12 0054D           BNEQ    74$
                57             05   D0 0054F           MOVL    #5, NUMBER_KIND                         ; 5678
                08             50   D1 00552 74$:     CMPL    R0, #8                                  ; 5680
                               03   12 00555           BNEQ    75$
                57             0B   D0 00557           MOVL    #11, NUMBER_KIND                        ; 5682
                02             50   D1 0055A 75$:     CMPL    R0, #2                                  ; 5684
                               03   12 0055D           BNEQ    76$
                57             0A   D0 0055F           MOVL    #10, NUMBER_KIND                        ; 5686
                5A             57   D0 00562 76$:     MOVL    NUMBER_KIND, BACKUP_NUMBER_KIND         ; 5688
                52 00000000'   EF   D0 00565 77$:     MOVL    CHARPTR, R2                             ; 5703
                50             62   9A 0056C           MOVZBL  (R2), R0
                53 00000000'EF40   DE 0056F           MOVAL   CHARTBL[R0], R3
04    AE              63       04   EF 00577           EXTZV   #4, #4, (R3), CLASS
                09 00000000G   00   91 0057D           CMPB    DBG$GB_LANGUAGE, #9                     ; 5704
                               0E   12 00584           BNEQ    78$
                2E             50   91 00586           CMPB    R0, #46                                 ; 5706
                               09   12 00589           BNEQ    78$
                2E        01   A2   91 0058B           CMPB    1(R2), #46
                               03   12 0058F           BNEQ    78$
                         04    AE   D4 00591           CLRL    CLASS                                   ; 5708
                   00000000'FF48   DF 00594 78$:     PUSHAL  @STATE_TABLE[STATE_INDEX]               ; 5717
                50             9E   9A 0059B           MOVZBL  @(SP)+, R0
                               0A   13 0059E           BEQL    79$
                      04       AE   50   D1 005A0     CMPL    R0, CLASS                               ; 5719
                               04   13 005A4           BEQL    79$
                               58   D6 005A6           INCL    STATE_INDEX                             ; 5721
                               EA   11 005A8           BRB     78$
```

```
                              00000000'FF48  DF 005AA 79$:   PUSHAL  @STATE_TABLE[STATE_INDEX]                ; 5723
 08   AE        9E        08             08  EF 005B1        EXTZV   #8, #8, @(SP)+, ACTION
                              00000000'FF48  DF 005B7        PUSHAL  @STATE_TABLE[STATE_INDEX]                ; 5724
      58        9E        10             10  EF 005BE        EXTZV   #16, #16, @(SP)+, STATE_INDEX
                11        01      08     AE  CF 005C3        CASEL   ACTION, #1, #17                          ; 5730
    0045      0040      003B           0127     005C8 80$:   .WORD   106$-80$,-                               ;
    008F      0059      004F           004A     005D0                82$-80$,-                                ;
    00D2      0024      00DD           00AD     005D8                83$-80$,-                                ;
    00B3      0072      005E           00C7     005E0                84$-80$,-                                ;
                        0054           0127     005E8                85$-80$,-                                ;
                                                                     86$-80$,-                                ;
                                                                     88$-80$,-                                ;
                                                                     94$-80$,-                                ;
                                                                     96$-80$,-                                ;
                                                                     101$-80$,-                               ;
                                                                     81$-80$,-                                ;
                                                                     100$-80$,-                               ;
                                                                     98$-80$,-                                ;
                                                                     89$-80$,-                                ;
                                                                     90$-80$,-                                ;
                                                                     97$-80$,-                                ;
                                                                     106$-80$,-                               ;
                                                                     87$-80$                                  ;
                              00000000'   EF  9F 005EC 81$:  PUSHAB  P.AXD                                    ; 5944
                                          01  DD 005F2       PUSHL   #1                                       ;
                              00028362    8F  DD 005F4       PUSHL   #164706                                  ;
                    00000000G  00         03  FB 005FA       CALLS   #3, LIB$SIGNAL                           ;
                                          51  11 00601       BRB     93$                                      ;
                          6E              52  D0 00603 82$:  MOVL    R2, BACKUP_DIGIT_PTR                     ; 5745
                                          4C  11 00606       BRB     93$                                      ;
                          57              06  D0 00608 83$:  MOVL    #6, NUMBER_KIND                          ; 5752
                                          47  11 0060B       BRB     93$                                      ;
                          6E              52  D0 0060D 84$:  MOVL    R2, BACKUP_DIGIT_PTR                     ; 5761
                                          3F  11 00610       BRB     92$                                      ; 5762
                          57              07  D0 00612 85$:  MOVL    #7, NUMBER_KIND                          ; 5769
                                          3D  11 00615       BRB     93$                                      ;
                          57              08  D0 00617 86$:  MOVL    #8, NUMBER_KIND                          ; 5775
                                          38  11 0061A       BRB     93$                                      ;
                          57              0F  D0 0061C 87$:  MOVL    #15, NUMBER_KIND                         ; 5781
                                          33  11 0061F       BRB     93$                                      ;
                          57              09  D0 00621 88$:  MOVL    #9, NUMBER_KIND                          ; 5787
                                          2E  11 00624       BRB     93$                                      ;
                          6E              52  D0 00626 89$:  MOVL    R2, BACKUP_DIGIT_PTR                     ; 5796
                          05              57  D1 00629       CMPL    NUMBER_KIND, #5                          ; 5797
                                          68  13 0062C       BEQL    99$                                      ;
                          0B              57  D1 0062E       CMPL    NUMBER_KIND, #11                         ; 5798
                                          63  13 00631       BEQL    99$                                      ;
                          0A              57  D1 00633       CMPL    NUMBER_KIND, #10                         ; 5799
                                          5E  13 00636       BEQL    99$                                      ;
                                          12  11 00638       BRB     91$                                      ; 5801
                          6E              52  D0 0063A 90$:  MOVL    R2, BACKUP_DIGIT_PTR                     ; 5811
                          05              57  D1 0063D       CMPL    NUMBER_KIND, #5                          ; 5812
                                          0F  13 00640       BEQL    92$                                      ;
                          0B              57  D1 00642       CMPL    NUMBER_KIND, #11                         ; 5813
                                          0A  13 00645       BEQL    92$                                      ;
                          0A              57  D1 00647       CMPL    NUMBER_KIND, #10                         ; 5814
                                          05  13 0064A       BEQL    92$                                      ;
```

```
                                5A              0E  D0 0064C 91$:    MOVL    #14, BACKUP_NUMBER_KIND                 ; 5816
                                                03  11 0064F        BRB     93$                                     ;
                                5A              06  D0 00651 92$:    MOVL    #6, BACKUP_NUMBER_KIND                  ; 5818
                                              0098  31 00654 93$:    BRW     106$                                    ; 5730
                                57              5A  D0 00657 94$:    MOVL    BACKUP_NUMBER_KIND, NUMBER_KIND         ; 5831
                00000000'       EF              6E  01 C1 0065A      ADDL3   #1, BACKUP_DIGIT_PTR, CHARPTR           ; 5832
                59 00000000'    EF              56  C3 00662        SUBL3   STARTPTR, CHARPTR, TOKENLEN             ; 5833
                   000000FF     8F              59  D1 0066A 95$:    CMPL    TOKENLEN, #255                          ; 5834
                                                5E  14 00671        BGTR    104$                                    ;
                                                69  11 00673        BRB     105$                                    ; 5835
                59              52              56  C3 00675 96$:    SUBL3   STARTPTR, R2, TOKENLEN                  ; 5847
                                                EF  11 00679        BRB     95$                                     ; 5848
                                05              57  D1 0067B 97$:    CMPL    NUMBER_KIND, #5                         ; 5861
                                                F5  13 0067E        BEQL    96$                                     ;
                                0B              57  D1 00680        CMPL    NUMBER_KIND, #11                        ; 5862
                                                F0  13 00683        BEQL    96$                                     ;
                                0A              57  D1 00685        CMPL    NUMBER_KIND, #10                        ; 5863
                                                EB  13 00688        BEQL    96$                                     ;
                                57              0E  D0 0068A        MOVL    #14, NUMBER_KIND                        ; 5865
                                                E6  11 0068D        BRB     96$                                     ; 5867
                                10 00000000'    EF  D1 0068F 98$:    CMPL    EXPRESSION_RADIX, #16                   ; 5889
                                                57  13 00696 99$:    BEQL    106$                                    ;
                                                0B  11 00698        BRB     101$                                    ; 5892
                                08              63  E8 0069A 100$:   BLBS    (R3), 101$                              ; 5909
                04              63              01  E0 0069D        BBS     #1, (R3), 101$                          ; 5910
                09              63              02  E1 006A1        BBC     #2, (R3), 102$                          ; 5911
                   00000000'    EF              56  D0 006A5 101$:   MOVL    STARTPTR, CHARPTR                       ; 5914
                                                4A  11 006AC        BRB     107$                                    ; 5913
                                05              57  D1 006AE 102$:   CMPL    NUMBER_KIND, #5                         ; 5918
                                                0D  13 006B1        BEQL    103$                                    ;
                                0B              57  D1 006B3        CMPL    NUMBER_KIND, #11                        ; 5919
                                                08  13 006B6        BEQL    103$                                    ;
                                0A              57  D1 006B8        CMPL    NUMBER_KIND, #10                        ; 5920
                                                03  13 006BB        BEQL    103$                                    ;
                                57              0E  D0 006BD        MOVL    #14, NUMBER_KIND                        ; 5922
            59 00000000'        EF              56  C3 006C0 103$:   SUBL3   STARTPTR, CHARPTR, TOKENLEN             ; 5924
               000000FF         8F              59  D1 006C8        CMPL    TOKENLEN, #255                          ; 5925
                                                0D  15 006CF        BLEQ    105$                                    ;
                                000289C2        8F  DD 006D1 104$:   PUSHL   #166338                                ;
                   00000000G    00              01  FB 006D7        CALLS   #1, LIB$SIGNAL                          ;
                           14   AE              59  90 006DE 105$:   MOVB    TOKENLEN, TOKENBUFFER                   ; 5926
          15    AE              66              59  28 006E2        MOVC3   TOKENLEN, (STARTPTR), TOKENBUFFER+1     ; 5927
                                                14  AE  9F 006E7    PUSHAB  TOKENBUFFER                             ; 5928
                                                57  DD 006EA        PUSHL   NUMBER_KIND                             ;
                                              06EC  31 006EC        BRW     202$                                    ;
                                00000000'       EF  D6 006EF 106$:   INCL    CHARPTR                                 ; 5951
                                              FE6D  31 006F5        BRW     77$                                     ; 5695
                                57 00000000'    EF  D0 006F8 107$:   MOVL    CHARPTR, R7                             ; 5963
                                50              67  9A 006FF        MOVZBL  (R7), R0                                ;
                                03 00000000'EF40    E8 C0702        BLBS    CHARTBL[R0], 108$                       ;
                                              0147  31 0070A        BRW     126$                                    ;
                                                57  D7 0070D 108$:   DECL    ENDPTR                                  ; 5974
                                51 00000000'    EF  D0 0070F        MOVL    CHARPTR, R1                             ; 5977
                                50              61  9A 00716 109$:   MOVZBL  (R1), R0                                ;
                                00000000'EF40       DF 00719        PUSHAL  CHARTBL[R0]                             ;
                   0E           9E              02  E1 00720        BBC     #2, @(SP)+, 110$                        ;
                                57              51  D0 00724        MOVL    R1, ENDPTR                              ; 5980
```

```
                        00000000'EF40   DF 00727          PUSHAL  CHARTBL[R0]                                                    ; 5981
            26              9E          01 E1 0072E      BBC     #1, @(SP)+, 111$
                        00000000'   EF  D6 00732 110$:   INCL    CHARPTR                                                        ; 5984
                        51 00000000'    EF  D0 00738     MOVL    CHARPTR, R1                                                    ; 5985
                        50              61  9A 0073F     MOVZBL  (R1), R0
                        00000000'EF40   DF 00742         PUSHAL  CHARTBL[R0]
            C9              9E          01 E0 00749      BBS     #1, @(SP)+, 109$
                        00000000'EF40   DF 0074D         PUSHAL  CHARTBL[R0]                                                    ; 5986
            BE              9E          02 E0 00754      BBS     #2, @(SP)+, 109$
                00000000'   EF      01  A7  9E 00758 111$:   MOVAB   1(R7), CHARPTR                                          ; 5990
            59 00000000'    EF          56  C3 00760     SUBL3   STARTPTR, CHARPTR, TOKENLEN                                    ; 5995
               000000FF     8F          59  D1 00768     CMPL    TOKENLEN, #255                                                 ; 5996
                                        0D  15 0076F     BLEQ    112$
                        00028982    8F  DD  00771        PUSHL   #166274
                00000000G   00          01  FB 00777    CALLS   #1, LIB$SIGNAL
            15  AE              66      59  28 0077E 112$:   MOVC3   TOKENLEN, (STARTPTR), TOKENBUFFER+1                       ; 5997
                    14          AE      59  90 00783     MOVB    TOKENLEN, TOKENBUFFER                                          ; 5998
                                3B  00000000'   EF  E9 00787 BLBC    CASING_SIGNIFICANT, 115$                                   ; 6010
                00000000G   00          56  D1 0078E     CMPL    STARTPTR, DBG$GL_UPCASE_COMMAND_PTR                            ; 6013
                                        09  19 00795     BLSS    113$
                00000000G   00          56  D1 00797     CMPL    STARTPTR, DBG$GL_UPCASE_COMMAND_PTR+4                          ; 6014
                                        15  15 0079E     BLEQ    114$
                        00000000'   EF  9F  007A0 113$:  PUSHAB  P.AXE                                                          ; 6016
                                        01  DD 007A6     PUSHL   #1
                        00028362    8F  DD  007A8        PUSHL   #164706
                00000000G   00          03  FB 007AE    CALLS   #3, LIB$SIGNAL
            50                  56  00000000G   00  C3 007B5 114$:   SUBL3   DBG$GL_UPCASE_COMMAND_PTR, STARTPTR, R0          ; 6018
                                50  00000000G   00  C0 007BD  ADDL2   DBG$GL_ORIG_COMMAND_PTR, NEW_STARTPTR                   ; 6019
            15  AE              60      59  28 007C4     MOVC3   TOKENLEN, (NEW_STARTPTR), TOKENBUFFER+1                       ; 6020
                                        5A  D4 007C9 115$:   CLRL    BEST_TOKEN_FOUND                                       ; 6028
                                55  00000000'   EF  D0 007CB  MOVL    IDENT_OPERATOR_TABLE, R5                               ; 6029
                                54              01  CE 007D2  MNEGL   #1, I
                                                32  11 007D5  BRB     121$
                                50  00000000'   EF  9E 007D7 116$:   MOVAB   TABLEBASE, R0                                   ; 6031
                58                  50          6544    C1 007DE  ADDL3   (R5)[I], R0, TOKEN
                                    50      0C      A8  9A 007E3  MOVZBL  12(TOKEN), R0                                       ; 6032
            59      00          0D  A8          50  2D 007E7     CMPC5   R0, 13(TOKEN), #0, TOKENLEN, TOKENBUFFER+1             ; 6033
                                            15  AE      007ED
                                            18  12 007EF     BNEQ    121$
                                    0C      0C  AE  E9 007F1  BLBC    12(SP), 119$                                           ; 6036
                                    02          68  91 007F5  CMPB    (TOKEN), #2                                            ; 6037
                                                03  12 007F8  BNEQ    118$
                                            0549    31 007FA 117$:   BRW     193$
                                    05      0C      AE  E8 007FD 118$:   BLBS    12(SP), 120$                                   ; 6038
                                    02          68  91 00801 119$:   CMPB    (TOKEN), #2                                    ; 6039
                                                F4  12 00804     BNEQ    117$
                                    5A          58  D0 00806 120$:   MOVL    TOKEN, BEST_TOKEN_FOUND                        ; 6044
                C9                  54      FC  A5  F2 00809 121$:   AOBLSS  -4(R5), I, T16$                                ; 6029
                                                5A  D5 0080E     TSTL    BEST_TOKEN_FOUND                                       ; 6050
                                                36  12 00810     BNEQ    124$
                                                5A  D4 00812     CLRL    BEST_TOKEN_FOUND                                       ; 6057
                                55  00000000'   EF  D0 00814  MOVL    BIF_TABLE, R5                                          ; 6058
                                54              01  CE 0081B  MNEGL   #1, I
                                                21  11 0081E  BRB     123$
                                50  00000000'   EF  9E 00820 122$:   MOVAB   TABLEBASE, R0                                   ; 6060
                58                  50          6544    C1 00827  ADDL3   (R5)[I], R0, TOKEN
                                    50      08      A8  9A 0082C  MOVZBL  8(TOKEN), R0                                       ; 6061
```

```
                59  00      09  A8          50  2D 00830          CMPC5   R0, 9(TOKEN), #0, TOKENLEN, TOKENBUFFER+1  ; 6062
                                15  AE          00836
                                    07  12  00838          BNEQ    123$
                                BC  0C  AE  E8  0083A          BLBS    12(SP), 117$                                ; 6064
                                5A      58  D0  0083E          MOVL    TOKEN, BEST_TOKEN_FOUND                     ; 6068
                                DA      54  FC  A5  F2  00841 123$:    AOBLSS  -4(R5), I, T22$                     ; 6058
                                            5A  D5  00846          TSTL    BEST_TOKEN_FOUND                    ; 6071
                                            03  13  00848 124$:    BEQL    125$
                                          01BF  31  0084A          BRW     154$
                                            59  D5  0084D 125$:    TSTL    TOKENLEN                            ; 6078
                                            03  13  0084F          BEQL    126$
                                          0582  31  00851          BRW     201$
                                50 00000000'  FF  9A  00854 126$:    MOVZBL  @CHARPTR, R0                        ; 6088
                                   00000000'EF40  DF  0085B          PUSHAL  CHARTBL+1[R0]
                                03      9E      03  E0  00862          BBS     #3, @(SP)+, 127$
                                          00F1  31  00866          BRW     143$
                                50 00000000'  FF  9A  00869 127$:    MOVZBL  @CHARPTR, R0                        ; 6100
                                   00000000'EF40  DF  00870          PUSHAL  CHARTBL+1[R0]
                                08      9E      04  E1  00877          BBC     #4, @(SP)+, 128$
                                   00000000'  EF  D6  0087B          INCL    CHARPTR                             ; 6101
                                            E6  11  00881          BRB     127$
                                50 00000000'  FF  9A  00883 128$:    MOVZBL  @CHARPTR, R0                        ; 6103
                                   00000000'EF40  DF  0088A          PUSHAL  CHARTBL+1[R0]
                                08      9E      05  E1  00891          BBC     #5, @(SP)+, 129$
                                   00000000'  EF  D6  00895          INCL    CHARPTR                             ; 6104
                                            E6  11  0089B          BRB     128$
                                50 00000000'  FF  9A  0089D 129$:    MOVZBL  @CHARPTR, R0                        ; 6106
                                   00000000'EF40  DF  008A4          PUSHAL  CHARTBL+1[R0]
                                08      9E      C6  E1  008AB          BBC     #6, @(SP)+, 130$
                                   00000000'  EF  D6  008AF          INCL    CHARPTR                             ; 6107
                                            E6  11  008B5          BRB     129$
                                56 00000000'  EF  D1  008B7 130$:    CMPL    CHARPTR, STARTPTR                   ; 6109
                                            06  12  008BE          BNEQ    131$
                                   00000000'  EF  D6  008C0          INCL    CHARPTR
                             59 00000000'  EF          56  C3  008C6 131$:    SUBL3   STARTPTR, CHARPTR, TOKENLEN         ; 6114
                                000000FF   8F          59  D1  008CE          CMPL    TOKENLEN, #255                      ; 6115
                                                       0D  15  008D5          BLEQ    132$
                                   000289A2   8F          DD  008D7          PUSHL   #166306
                                00000000G  00          01  FB  008DD          CALLS   #1, LIB$SIGNAL
                                      14   AE          59  90  008E4 132$:    MOVB    TOKENLEN, TOKENBUFFER               ; 6116
                            15  AE          66          59  28  008E8          MOVC3   TOKENLEN, (STARTPTR), TOKENBUFFER+1 ; 6117
                                                        5A  D4  008ED          CLRL    BEST_TOKEN_FOUND                    ; 6126
                                55 00000000'  EF  D0  008EF          MOVL    OPCHAR_OPERATOR_TABLE, R5           ; 6127
                                54          01  CE  008F6          MNEGL   #1, I
                                            3A  11  008F9          BRB     139$
                                50 00000000'  EF  9E  008FB 133$:    MOVAB   TABLEBASE, R0                       ; 6129
                                58          50      6544  C1  00902          ADDL3   (R5)[I], R0, TOKEN
                                            04      01  A8  E8  00907          BLBS    1(TOKEN), 134$              ; 6130
                                            26      08  AC  E8  0090B          BLBS    ADDRESS_EXPRESSION, 139$
                                            50      0C  A8  9A  0090F 134$:    MOVZBL  12(TOKEN), R0               ; 6131
                59          00      0D  A8          50  2D  00913          CMPC5   R0, 13(TOKEN), #0, TOKENLEN, TOKENBUFFER+1  ; 6132
                                            15  AE          00919
                                                18  12  0091B          BNEQ    139$
                                            0C      0C  AE  E9  0091D          BLBC    12(SP), 137$                ; 6135
                                            02          68  91  00921          CMPB    (TOKEN), #2                 ; 6136
                                                        03  12  00924          BNEQ    136$
                                                      041D  31  00926 135$:    BRW     193$
```

```
                        05        0C    AE  E8 00929 136$:   BLBS    12(SP), 138$                          ; 6137
                        02              68  91 0092D 137$:   CMPB    (TOKEN), #2                           ; 6138
                                        F4  12 00930          BNEQ    135$                                  ;
                        5A              58  D0 00932 138$:   MOVL    TOKEN, BEST_TOKEN_FOUND               ; 6143
                54      FC              A5  F2 00935 139$:   AOBLSS  -4(R5), I, T33$                       ; 6127
                                        5A  D5 0093A          TSTL    BEST_TOKEN_FOUND                      ; 6155
                                        15  13 0093C          BEQL    142$                                  ;
                        03        08    AC  E8 0093E          BLBS    ADDRESS_EXPRESSION, 141$              ; 6156
                                  00C7  31  00942 140$:       BRW     154$                                  ;
                50                      66  9A 00945 141$:   MOVZBL  (STARTPTR), R0                       ; 6157
                    00000000'EF40       DF  00948          PUSHAL  CHARTBL+1[R0]                         ;
                                        9E  95 0094F          TSTB    @(SP)+                                ;
                                        EF  18 00951          BGEQ    140$                                  ;
        00000000'               56      D0 00953 142$:      MOVL    STARTPTR, CHARPTR                     ; 6165
                        03        08    AC  E8 0095A 143$:   BLBS    ADDRESS_EXPRESSION, 145$              ; 6176
                                  00B6  31  0095E 144$:       BRW     156$                                  ;
                50 00000000'            FF  9A 00961 145$:   MOVZBL  @CHARPTR, R0                          ;
                    00000000'EF40       DF  00968          PUSHAL  CHARTBL+1[R0]                         ;
                                        9E  95 0096F          TSTB    @(SP)+                                ;
                                        EB  18 00971          BGEQ    144$                                  ;
                                        51  D4 00973          CLRL    R1                                    ; 6182
                07 00000000G            00  91 00975          CMPB    DBG$GB_LANGUAGE, #7                   ;
                                        22  12 0097C          BNEQ    146$                                  ;
                                        51  D6 0097E          INCL    R1                                    ;
                2D 00000000'            FF  91 00980          CMPB    @CHARPTR, #45                         ; 6183
                                        17  12 00987          BNEQ    146$                                  ;
                50 00000000'            EF  D0 00989          MOVL    CHARPTR, R0                           ;
                3E              01      A0  91 00990          CMPB    1(R0), #62                            ;
                                        0A  12 00994          BNEQ    146$                                  ;
        00000000'               EF      02  C0 00996          ADDL2   #2, CHARPTR                           ; 6186
                                  021F  31  0099D          BRW     174$                                  ; 6187
                1B                      51  E9 009A0 146$:   BLBC    R1, 147$                              ; 6189
                2A 00000000'            FF  91 009A3          CMPB    @CHARPTR, #42                         ; 6190
                                        12  12 009AA          BNEQ    147$                                  ;
                0E        0C            AE  E9 009AC          BLBC    12(SP), 147$                          ; 6191
                    00000000'           EF  D6 009B0          INCL    CHARPTR                               ; 6194
                50 00000000'            EF  9E 009B6          MOVAB   C_INDIRECT_TOKEN, R0                  ; 6195
                                        04  009BD          RET                                           ;
                    00000000'           EF  D6 009BE 147$:   INCL    CHARPTR                               ; 6200
                                        5A  D4 009C4          CLRL    BEST_TOKEN_FOUND                      ; 6201
                54                      01  CE 009C6          MNEGL   #1, I                                 ; 6206
                                        35  11 009C9          BRB     153$                                  ;
                50 00000000'            EF  9E 009CB 148$:   MOVAB   TABLEBASE, R0                         ; 6204
        58              50 00000000'EF44    C1 009D2          ADDL3   ADDR_EXPR_OPTBL[I], R0, TOKEN         ;
                50        0C            A8  9A 009DB          MOVZBL  12(TOKEN), R0                         ; 6205
01      00      0D      A8              50  2D 009DF          CMPC5   R0, 13(TOKEN), #0, #1, (STARTPTR)     ; 6206
                                        66      009E5                                                     ;
                                        18  12 009E6          BNEQ    153$                                  ;
                0C        0C            AE  E9 009E8          BLBC    12(SP), 151$                          ; 6209
                02              68      91  009EC          CMPB    (TOKEN), #2                           ; 6210
                                        03  12 009EF          BNEQ    150$                                  ;
                                  0352  31  009F1 149$:       BRW     193$                                  ;
                05        0C            AE  E8 009F4 150$:   BLBS    12(SP), 152$                          ; 6211
                02              68      91  009F8 151$:   CMPB    (TOKEN), #2                           ; 6212
                                        F4  12 009FB          BNEQ    149$                                  ;
                5A                      58  D0 009FD 152$:   MOVL    TOKEN, BEST_TOKEN_FOUND               ; 6217
        C3              54 00000000'    EF  F2 00A00 153$:   AOBLSS  ADDR_EXPR_OPTBL-4, I, 148$            ; 6202
```

```
                                                 5A  D5 00A08          TSTL    BEST_TOKEN_FOUND                                    ; 6223
                                                 04  13 00A0A          BEQL    155$
                                       50        5A  D0 00A0C 154$:    MOVL    BEST_TOKEN_FOUND, R0
                                                 04  00A0F          RET
                                 00000000'       EF  56  D0 00A10 155$:    MOVL    STARTPTR, CHARPTR                                   ; 6224
                                      0A         00  00000000G  00  8F 00A17 156$:    CASEB   DBG$GB_LANGUAGE, #0, #10                    ; 6233
                                03C2        03C2        0019        03C2      00A1F 157$:    .WORD   203$-157$,-
                                00F9        03C2        03C2        03C2      00A27           158$-157$,-
                                            03C2        0267        01BC      00A2F           203$-157$,-
                                                                                              203$-157$,-
                                                                                              203$-157$,-
                                                                                              203$-157$,-
                                                                                              203$-157$,-
                                                                                              167$-157$,-
                                                                                              177$-157$,-
                                                                                              186$-157$,-
                                                                                              203$-157$
                                                        03A9  31 00A35          BRW     203$
                                       1B       08      AC  E9 00A38 158$:    BLBC    ADDRESS_EXPRESSION, 159$                            ; 6248
                                       17       0C      AE  E9 00A3C          BLBC    12(SP), 159$
                                       2E 00000000'     FF  91 00A40          CMPB    @CHARPTR, #46                                       ; 6249
                                                        0E  12 00A47          BNEQ    159$
                                          00000000'     EF  D6 00A49          INCL    CHARPTR                                             ; 6252
                                       50 00000000'     EF  9E 00A4F          MOVAB   FORTRAN_INDIRECT_TOKEN, R0                          ; 6253
                                                        04  00A56          RET
                                       50 00000000'     EF  D0 00A57 159$:    MOVL    CHARPTR, R0                                         ; 6260
                                                        50  D6 00A5E          INCL    R0
                                       50               60  9A 00A60          MOVZBL  (R0), R0
                                       08 00000000'EF   40  E9 00A63          BLBC    CHARTBL+1[R0], 160$
                                          00000000'     EF  D6 00A6B          INCL    CHARPTR                                             ; 6261
                                                        E4  11 00A71          BRB     159$
                                    00000000'           EF      02  C0 00A73 160$:    ADDL2   #2, CHARPTR                                 ; 6263
                                 55 00000000'           EF      56  C3 00A7A          SUBL3   STARTPTR, CHARPTR, R5                       ; 6268
                                 54                             01  CE 00A82          MNEGL   #1, I
                                                                20  11 00A85          BRB     162$
                                                    50 00000000'   EF  9E 00A87 161$:    MOVAB   TABLEBASE, R0                       ; 6266
                                 58                 50 00000000'EF44  C1 00A8E          ADDL3   FORTRAN_SPECIAL_OPTBL[I], R0, TOKEN
                                                    50       0C    A8  9A 00A97          MOVZBL  12(TOKEN), R0                       ; 6267
                          55     00        0D      A8                50  2D 00A9B          CMPC5   R0, 13(TOKEN), #0, R5, (STARTPTR)
                                                                      66      00AA1
                                                                      03  12 00AA2          BNEQ    162$
                                                                  029F    31 00AA4          BRW     193$
                                 D8                 54 00000000'   EF  F2 00AA7 162$:    AOBLSS  FORTRAN_SPECIAL_OPTBL-4, I, 161$    ; 6264
                                                    50 00000000'   EF  D0 00AAF          MOVL    PRIDTBL, R0                         ; 6277
                                          08    AE        FC      A0  D0 00AB6          MOVL    -4(R0), 8(SP)
                                                5A                  01  CE 00ABB          MNEGL   #1, I
                                                                    37  11 00ABE          BRB     164$
                                                    50 00000000'   EF  9E 00AC0 163$:    MOVAB   TABLEBASE, R0                       ; 6279
                                 57                 50 00000000'FF4A  C1 00AC7          ADDL3   @PRIDTBL[I], R0, PRID
                                                    51       08    A7  9A 00AD0          MOVZBL  8(PRID), R1                         ; 6280
                                 50 00000000'       EF              56  C3 00AD4          SUBL3   STARTPTR, CHARPTR, R0               ; 6281
                          50     00        09      A7                51  2D 00ADC          CMPC5   R1, 9(PRID), #0, R0, (STARTPTR)     ; 6280
                                                                      66      00AE2
                                                                      12  12 00AE3          BNEQ    164$
                                                    50       08    A7  9A 00AE5          MOVZBL  8(PRID), R0                         ; 6284
                                    15  AE        09      A7                50  28 00AE9          MOVC3   R0, 9(PRID), TOKENBUFFER+1
                                                14  AE        08    A7  90 00AEF          MOVB    8(PRID), TOKENBUFFER                ; 6285
```

```
                        02DF   31 00AF4          BRW     201$                                    ; 6286
C4              5A       08    AE F2 00AF7 164$:  AOBLSS  8(SP), I, 163$                          ; 6277
                03       0C    AE E9 00AFC        BLBC    12(SP), 166$                            ; 6294
                           02DE 31 00B00 165$:    BRW     203$                                    ;
                2E              66 91 00B03 166$:  CMPB    (STARTPTR), #46                         ;
                                F8 12 00B06        BNEQ    165$                                    ;
    00000000'   EF          01  A6 9E 00B08        MOVAB   1(R6), CHARPTR                          ; 6297
                50 00000000'    EF 9E 00B10        MOVAB   FORTRAN_DOT_TOKEN, R0                   ; 6298
                                04 00B17           RET                                             ;
                26 00000000'    FF 91 00B18 167$:  CMPB    @CHARPTR, #38                           ; 6316
                                31 12 00B1F        BNEQ    170$                                    ;
                   00000000'    EF D6 00B21        INCL    CHARPTR                                 ; 6319
                26 00000000'    FF 91 00B27        CMPB    @CHARPTR, #38                           ; 6320
                                0E 12 00B2E        BNEQ    168$                                    ;
                   00000000'    EF D6 00B30        INCL    CHARPTR                                 ; 6323
                50 00000000'    EF 9E 00B36        MOVAB   C_AND_TOKEN, R0                         ; 6324
                                04 00B3D           RET                                             ;
                08       0C     AE E9 00B3E 168$:  BLBC    12(SP), 169$                            ; 6327
                50 00000000'    EF 9E 00B42        MOVAB   C_ADDR_OF_TOKEN, R0                     ;
                                04 00B49           RET                                             ;
                50 00000000'    EF 9E 00B4A 169$:  MOVAB   C_BIT_AND_TOKEN, R0                     ; 6328
                                04 00B51           RET                                             ;
                2B 00000000'    FF 91 00B52 170$:  CMPB    @CHARPTR, #43                           ; 6335
                                2A 12 00B59        BNEQ    172$                                    ;
                   00000000'    EF D6 00B5B        INCL    CHARPTR                                 ; 6338
                2B 00000000'    FF 91 00B61        CMPB    @CHARPTR, #43                           ; 6339
                                13 12 00B68        BNEQ    171$                                    ;
                   00000000'    EF D6 00B6A        INCL    CHARPTR                                 ; 6342
                   00028F78     8F DD 00B70        PUSHL   #167800                                 ; 6357
    00000000G   00              01 FB 00B76        CALLS   #1, LIB$SIGNAL                          ;
                50 00000000'    EF 9E 00B7D 171$:  MOVAB   C_ADD_TOKEN, R0                         ; 6361
                                04 00B84           RET                                             ;
                2D 00000000'    FF 91 00B85 172$:  CMPB    @CHARPTR, #45                           ; 6368
                                75 12 00B8C        BNEQ    179$                                    ;
                   00000000'    EF D6 00B8E        INCL    CHARPTR                                 ; 6371
                2D 00000000'    FF 91 00B94        CMPB    @CHARPTR, #45                           ; 6372
                                13 12 00B9B        BNEQ    173$                                    ;
                   00000000'    EF D6 00B9D        INCL    CHARPTR                                 ; 6375
                   00028F78     8F DD 00BA3        PUSHL   #167800                                 ; 6389
    00000000G   00              01 FB 00BA9        CALLS   #1, LIB$SIGNAL                          ;
                3E 00000000'    FF 91 00BB0 173$:  CMPB    @CHARPTR, #62                           ; 6393
                                0E 12 00BB7        BNEQ    175$                                    ;
                   00000000'    EF D6 00BB9        INCL    CHARPTR                                 ; 6396
                50 00000000'    EF 9E 00BBF 174$:  MOVAB   C_ARROW_TOKEN, R0                       ; 6397
                                04 00BC6           RET                                             ;
                08       0C     AE E9 00BC7 175$:  BLBC    12(SP), 176$                            ; 6400
                50 00000000'    EF 9E 00BCB        MOVAB   C_MINUS_TOKEN, R0                       ; 6402
                                04 00BD2           RET                                             ; 6404
                50 00000000'    EF 9E 00BD3 176$:  MOVAB   C_SUB_TOKEN, R0                         ;
                                04 00BDA           RET                                             ;
                2A 00000000'    FF 91 00BDB 177$:  CMPB    @CHARPTR, #42                           ; 6414
                                1F 12 00BE2        BNEQ    179$                                    ;
                03       0C     AE E8 00BE4        BLBS    12(SP), 178$                            ; 6420
                           008D 31 00BE8          BRW     185$                                    ;
                   00000000'    EF D6 00BEB 178$:  INCL    CHARPTR                                 ; 6423
                51 00000000'    EF D0 00BF1        MOVL    CHARPTR, R1                             ; 6424
                50              61 9A 00BF8        MOVZBL  (R1), R0                                ;
```

```
                03 00000000'EF40   E8 00BFB          BLBS    CHARTBL[R0], 180$
                         01DB      31 00C03 179$:     BRW     203$
                57        FE   A1   9E 00C06 180$:     MOVAB   -2(R1), ENDPTR                          ;  6435
                51 00000000'   EF   D0 00C0A          MOVL    CHARPTR, R1                             ;  6438
                50             61   9A 00C11 181$:     MOVZBL  (R1), R0
                   00000000'EF40   DF 00C14          PUSHAL  CHARTBL[R0]
          0E              9E        02 E1 00C1B          BBC     #2, @(SP)+, 182$
                57              51   D0 00C1F          MOVL    R1, ENDPTR                              ;  6441
                   00000000'EF40   DF 00C22          PUSHAL  CHARTBL[R0]                             ;  6442
          26              9E        01 E1 00C29          BBC     #1, @(SP)+, 183$
                   00000000'   EF   D6 00C2D 182$:     INCL    CHARPTR                                 ;  6445
                51 00000000'   EF   D0 00C33          MOVL    CHARPTR, R1                             ;  6446
                50             61   9A 00C3A          MOVZBL  (R1), R0
                   00000000'EF40   DF 00C3D          PUSHAL  CHARTBL[R0]
          C9              9E        01 E0 00C44          BBS     #1, @(SP)+, 181$
                   00000000'EF40   DF 00C48          PUSHAL  CHARTBL[R0]                             ;  6447
          BE              9E        02 E0 00C4F          BBS     #2, @(SP)+, 181$
   00000000'   EF        01   A7   9E 00C53 183$:     MOVAB   1(R7), CHARPTR                          ;  6451
59 00000000'   EF             56   C3 00C5B          SUBL3   STARTPTR, CHARPTR, TOKENLEN             ;  6456
   000000FF    8F             59   D1 00C63          CMPL    TOKENLEN, #255                          ;  6457
                            03   14 00C6A          BGTR    184$
                         F843   31 00C6C          BRW     60$
                   00028982    8F   DD 00C6F 184$:     PUSHL   #166274
                         F833   31 00C75          BRW     59$
                   00000000'   EF   D6 00C78 185$:     INCL    CHARPTR                                 ;  6468
                50 00000000'   EF   9E 00C7E          MOVAB   RPG_MULTIPLY_TOKEN, R0                  ;  6469
                            04   00C85          RET
                27 00000000'   FF   91 00C86 186$:     CMPB    @CHARPTR, #39                           ;  6485
                            03   13 00C8D          BEQL    187$
                         014F   31 00C8F          BRW     203$
                03        0C   AE   E9 00C92 187$:     BLBC    12(SP), 188$                            ;  6494
                         0108   31 00C96          BRW     198$
                            08   DD 00C99 188$:     PUSHL   #8                                      ;  6500
      00000000G    00             01   FB 00C9B          CALLS   #1, DBG$GET_TEMPMEM
                58              50   D0 00CA2          MOVL    R0, TOKEN
          68 00000000'   EF        0E   28 00CA5          MOVC3   #14, ADA_TICK_TOKEN, (TOKEN)            ;  6501
                57              01   D0 00CAD          MOVL    #1, INDEX                               ;  6505
                50 00000000'   EF   9E 00CB0 189$:     MOVAB   TABLEBASE, R0                           ;  6508
          5B              50 00000000'EF47   C1 00CB7          ADDL3   ADA_TICK_TABLE[INDEX], R0, NAMEPTR
                51              6B   9A 00CC0          MOVZBL  (NAMEPTR), R1                           ;  6510
                50              6B   9A 00CC3          MOVZBL  (NAMEPTR), R0
                54 00000000'   EF   D0 00CC6          MOVL    CHARPTR, R4
50        00        01   AB   51   2D 00CCD          CMPC5   R1, 1(NAMEPTR), #0, R0, 1(R4)
                         01   A4      00CD3
                            73   12 00CD5          BNEQ    194$
                50              6B   9A 00CD7          MOVZBL  (NAMEPTR), R0                           ;  6513
      00000000'   EF        01 A440   9E 00CDA          MOVAB   1(R4)[R0], CHARPTR
            06   A8             57   B0 00CE3          MOVW    INDEX, 6(TOKEN)                         ;  6514
                50              6B   9A 00CE7          MOVZBL  (NAMEPTR), R0                           ;  6515
                                50   D6 00CEA          INCL    R0
            0C   A8             50   90 00CEC          MOVB    R0, 12(TOKEN)
            0C   AE             6B   9A 00CF0          MOVZBL  (NAMEPTR), 12(SP)                       ;  6516
                5A              50   D0 00CF4          MOVL    R0, R10                                 ;  6517
                59        0D    A8   9E 00CF7          MOVAB   13(R8), R9
5A        20 00000000'   EF        01   2C 00CFB          MOVC5   #1, P.AXF, #32, R10, (R9)
                                69      00D04
                                0C   18 00D05          BGEQ    190$
```

```
                                        59 D6 00D07          INCL    R9                                                      ;
                                        5A D7 00D09          DECL    R10                                                     ;
        5A      20      01      AB      0C AE 2C 00D0B       MOVC5   12(SP), 1(NAMEPTR), #32, R10, (R9)                      ;
                                        69    00D12                                                                          ;
                                50 00000000' FF 9A 00D13 190$:  MOVZBL  @CHARPTR, R0                                        ; 6527
                                   00000000'EF40 DF 00D1A       PUSHAL  CHARTBL+2[R0]                                       ;
                        08              9E    01 E1 00D21       BBC     #1, @(SP)+, 191$                                    ;
                                   00000000' EF D6 00D25        INCL    CHARPTR                                             ; 6528
                                             E6 11 00D2B        BRB     190$                                                ;
                                28 00000000' FF 91 00D2D 191$:  CMPB    @CHARPTR, #40                                       ; 6530
                                             0C 12 00D34        BNEQ    192$                                                ;
                        01      A8              08 88 00D36        BISB2   #8, 1(TOKEN)                                        ; 6533
                                   00000000' EF D6 00D3A        INCL    CHARPTR                                             ; 6534
                                             04 11 00D40        BRB     193$                                                ; 6530
                        01      A8              08 8A 00D42 192$:  BICB2   #8, 1(TOKEN)                                        ; 6537
                                50              58 D0 00D46 193$:  MOVL    TOKEN, R0                                           ; 6539
                                                04 00D49           RET                                                         ;
        FF60    57              01              09 F1 00D4A 194$:  ACBL    #9, #1, INDEX, 189$                                 ; 6505
                        14      AE      2701    8F B0 00D50        MOVW    #9985, TOKENBUFFER                                  ; 6546
                                   00000000' EF D6 00D56 195$:  INCL    CHARPTR                                             ; 6548
                                51 00000000' FF 9A 00D5C        MOVZBL  @CHARPTR, R1                                        ; 6549
                                50 00000000'EF41 DE 00D63       MOVAL   CHARTBL[R1], R0                                     ;
                                08              60 E8 00D6B        BLBS    (R0), 196$                                          ;
                        04              60      01 E0 00D6E        BBS     #1, (R0), 196$                                      ; 6550
                        19              60      02 E1 00D72        BBC     #2, (R0), 197$                                      ; 6551
                                0D              51 91 00D76 196$:  CMPB    R1, #13                                             ; 6553
                                                14 13 00D79        BEQL    197$                                                ;
                                20      14      AE 91 00D7B        CMPB    TOKENBUFFER, #32                                    ;
                                                0E 1E 00D7F        BGEQU   197$                                                ;
                                        14      AE 96 00D81        INCB    TOKENBUFFER                                         ; 6555
                                50      14      AE 9A 00D84        MOVZBL  TOKENBUFFER, R0                                     ; 6556
                            14 AE40             51 90 00D88        MOVB    R1, TOKENBUFFER[R0]                                 ;
                                                C7 11 00D8D        BRB     195$                                                ; 6557
                                        14      AE 9F 00D8F 197$:  PUSHAB  TOKENBUFFER                                         ; 6560
                                                01 DD 00D92        PUSHL   #1                                                  ;
                                    00028D30    8F DD 00D94        PUSHL   #167216                                             ;
                    00000000G   00              03 FB 00D9A        CALLS   #3, L!B$SIGNAL                                      ;
                                50 00000000'    EF D0 00DA1 198$:  MOVL    CHARPTR, R0                                         ; 6567
                                0D          01  A0 91 00DA8        CMPB    1(R0), #13                                          ;
                                                06 13 00DAC        BEQL    199$                                                ;
                                27          02  A0 91 00DAE        CMPB    2(R0), #39                                          ;
                                                0D 13 00DB2        BEQL    200$                                                ;
                                    00028992    8F DD 00DB4 199$:  PUSHL   #166290                                             ; 6569
                    00000000G   00              01 FB 00DBA        CALLS   #1, LIB$SIGNAL                                      ;
                            14  AE              03 90 00DC1 200$:  MOVB    #3, TOKENBUFFER                                     ; 6571
    15  AE              18          00 00000000' FF F0 00DC5        INSV    @CHARPTR, #0, #24, TOKENBUFFER+1                    ; 6572
                    00000000'   EF              03 C0 00DCF        ADDL2   #3, CHARPTR                                         ; 6573
                                        14      AE 9F 00DD6 201$:  PUSHAB  TOKENBUFFER                                         ; 6574
                                                01 DD 00DD9        PUSHL   #1                                                  ;
                                0000V   CF          02 FB 00DDB 202$:  CALLS   #2, CREATE_OPERAND_TOKEN                            ;
                                                04 00DE0           RET                                                         ;
                    00000000'   EF              56 D0 00DE1 203$:  MOVL    STARTPTR, CHARPTR                                   ; 6591
                                        14      AE 94 00DE8        CLRB    TOKENBUFFER                                         ; 6592
                                                50 D4 00DEB        CLRL    I                                                   ; 6595
                                0D 00000000'FF40 91 00DED 204$:  CMPB    @CHARPTR[I], #13                                    ;
                                                11 13 00DF5        BEQL    205$                                                ;
                            15 AE40 00000000'FF40 90 00DF7        MOVB    @CHARPTR[I], TOKENBUFFER+1[I]                       ; 6596
```

```
                        14   AE  96 00E01          INCB    TOKENBUFFER                     ; 6597
                E5           14  F3 00E04          AOBLEQ  #20, I, 204$                    ; 6593
                        14   AE  9F 00E08 205$:     PUSHAB  TOKENBUFFER                     ; 6600
                             01  DD 00E0B          PUSHL   #1                              ;
                 000289E2    8F  DD 00E0D          PUSHL   #166370                         ;
    00000000G   00           03  FB 00E13          CALLS   #3, LIB$SIGNAL                  ;
                             50  D4 00E1A          CLRL    R0                              ; 6601
                                 04 00E1C          RET                                     ; 6603

; Routine Size:  3613 bytes,    Routine Base:  DBG$CODE + 0A9E
```

```
;  6494          6604  1 GLOBAL ROUTINE DBG$PARSER_SET_LANGUAGE(LANGUAGE): NOVALUE =
;  6495          6605  1 !
;  6496          6606  1 ! FUNCTION
;  6497          6607  1 !       This routine sets up the parse tables used by the Parser and Lexical
;  6498          6608  1 !       Scanner for a specified language.  It is called during the processing
;  6499          6609  1 !       of the SET LANGUAGE command.
;  6500          6610  1 !
;  6501          6611  1 !       Specificially, this routine sets up the Character Table CHARTBL to
;  6502          6612  1 !       have the values appropriate for the specified language.  It does so
;  6503          6613  1 !       by initializing CHRTBL to have the values appropriate for language
;  6504          6614  1 !       UNKNOWN and then changing selected character characteristics as speci-
;  6505          6615  1 !       fied in the Character Exception Table for the specified language.
;  6506          6616  1 !       It also sets up pointers to the Identifier Operator Table, the Opera-
;  6507          6617  1 !       tor Character Operator Table, the Number Scanner State Table, the
;  6508          6618  1 !       Primary Parser State Table, and the Subscript Terminator Table for
;  6509          6619  1 !       the specified language. It calls the routine DBG$EVALOP_SET_LANGUAGE
;  6510          6620  1 !       to set up the Operator Information Tables.
;  6511          6621  1 !
;  6512          6622  1 ! INPUTS
;  6513          6623  1 !       LANGUAGE - The language code for the language being SET.
;  6514          6624  1 !
;  6515          6625  1 ! OUTPUTS
;  6516          6626  1 !       NONE
;  6517          6627  1 !
;  6518          6628  1
;  6519          6629  2     BEGIN
;  6520          6630  2
;  6521          6631  2     LOCAL
;  6522          6632  2         CEPTR: REF CE_ENTRY,            ! Pointer to Character Exception Table
;  6523          6633  2                                         !      entry for one character
;  6524          6634  2         CETBL: REF VECTOR[,LONG],       ! Pointer to Character Exception Table
;  6525          6635  2                                         !      for the language being SET
;  6526          6636  2         PTR: REF VECTOR[,LONG];         ! Pointer to table of language table
;  6527          6637  2                                         !      pointers for language being SET
;  6528          6638  2
;  6529          6639  2
;  6530          6640  2
;  6531          6641  2     ! Set PTR to point to the table of pointers for this language which point
;  6532          6642  2     ! to the various language-specific tables.  Use language UNKNOWN if we do
;  6533          6643  2     ! not recognize the language code.
;  6534          6644  2     !
;  6535          6645  2     IF (.LANGUAGE GEQ DBG$K_MIN_LANGUAGE) AND
;  6536          6646  3        (.LANGUAGE LEQ DBG$K_MAX_LANGUAGE)
;  6537          6647  2     THEN
;  6538          6648  2         PTR = .LANGUAGE_TABLE_PTRS[.LANGUAGE] + TABLEBASE
;  6539          6649  2
;  6540          6650  2     ELSE
;  6541          6651  2         PTR = .LANGUAGE_TABLE_PTRS[DBG$K_UNKNOWN] + TABLEBASE;
;  6542          6652  2
;  6543          6653  2
;  6544          6654  2     ! Set up the Character Table CHARTBL for this language.  Also make the
;  6545          6655  2     ! global pointer DBG$GL_CHARTBL point to the character table--this pointer
;  6546          6656  2     ! is used in DBGSOURCE by the SEARCH command.
;  6547          6657  2     !
;  6548          6658  2     DBG$GL_CHARTBL = CHARTBL;
;  6549          6659  2     CETBL = .PTR[0] + TABLEBASE;
;  6550          6660  2     CH$MOVE(256*%UPVAL, BASE_CHARACTER_TABLE, CHARTBL);
```

```
;  6551          6661  2     INCR I FROM 0 TO .CETBL[-1] - 1 DO
;  6552          6662  3         BEGIN
;  6553          6663  3         CEPTR = CETBL[.I];
;  6554          6664  3         CHARTBL[.CEPTR[CE_CHAR], CHRTBL$L_WHOLE_ENTRY] = .CEPTR[CE_BITS];
;  6555          6665  2         END;
;  6556          6666  2
;  6557          6667  2
;  6558          6668  2     ! Set up the various parse table pointers to the tables for this language.
;  6559          6669  2     !
;  6560          6670  2     IDENT_OPERATOR_TABLE = .PTR[1] + TABLEBASE;
;  6561          6671  2     OPCHAR_OPERATOR_TABLE = .PTR[2] + TABLEBASE;
;  6562          6672  2     STATE_TABLE = .PTR[3] + TABLEBASE;
;  6563          6673  2     PRIMARY_TABLE = .PTR[4] + TABLEBASE;
;  6564          6674  2     SUBSCRIPT_TERM_TBL = .PTR[5] + TABLEBASE;
;  6565          6675  2     PRIDTBL = .PTR[6] + TABLEBASE;
;  6566          6676  2     BIF_TABLE = .PTR[7] + TABLEBASE;
;  6567          6677  2     MULTIPLE_SUBSCR = .PTR[8];
;  6568          6678  2     ENFORCE_RECORD = .PTR[9];
;  6569          6679  2     CASING_SIGNIFICANT = .PTR[10];
;  6570          6680  2     COMPONENTS_IN_PATHNAME = .PTR[11];
;  6571          6681  2     INCOMPLETE_QUAL = .PTR[12];
;  6572          6682  2
;  6573          6683  2     ! Initialize the Operator Evaluation tables and the Print tables for
;  6574          6684  2     ! the current language.
;  6575          6685  2     !
;  6576          6686  2     DBG$EVALOP_SET_LANGUAGE (.LANGUAGE);
;  6577          6687  2     DBG$PRINT_SET_LANGUAGE (.LANGUAGE);
;  6578          6688  2     RETURN;
;  6579          6689  2
;  6580          6690  1     END;


                                                07FC 00000          .ENTRY  DBG$PARSER_SET_LANGUAGE, Save R2,R3,R4,R5,- ; 6604
                                                                            R6,R7,R8,R9,R10
                                    5A 00000000'  EF   9E 00002     MOVAB   TABLEBASE, R10
                                    59 00000000'  EF   9E 00009     MOVAB   CHARTBL, R9
                                    58        04  AC   D0 00010     MOVL    LANGUAGE, R8                                ; 6645
                                                  11   19 00014     BLSS    1$
                                    0A            58   D1 00016     CMPL    R8, #10                                     ; 6646
                                                  0C   14 00019     BGTR    1$
                                    50            6A   9E 0001B     MOVAB   TABLEBASE, R0                               ; 6648
                              56    50      2F91 CA48  C1 0001E     ADDL3   LANGUAGE_TABLE_PTRS[R8], R0, PTR
                                                  09   11 00025     BRB     2$
                                    50            6A   9E 00027 1$: MOVAB   TABLEBASE, R0                               ; 6651
                              56    50      2FB9  CA   C1 0002A     ADDL3   LANGUAGE_TABLE_PTRS+40, R0, PTR
                    00000000'   EF                69   9E 00030 2$: MOVAB   CHARTBL, DBG$GL_CHARTBL                     ; 6658
                                    50            6A   9E 00037     MOVAB   TABLEBASE, R0                               ; 6659
                              57    50            66   C1 0003A     ADDL3   (PTR), R0, CETBL
                              69      0591  CA      0400  8F   28 0003E     MOVC3   #1024, BASE_CHARACTER_TABLE, CHARTBL    ; 6660
                                    51            01   CE 00046     MNEGL   #1, I                                       ; 6661
                                                  0E   11 00049     BRB     4$
                                    52          6741   DE 0004B 3$: MOVAL   (CETBL)[I], CEPTR                          ; 6663
                                    50        03  A2   9A 0004F     MOVZBL  3(CEPTR), R0                                ; 6664
                    6940            62            18   00   EF 00053     EXTZV   #0, #24, (CEPTR), CHARTBL[R0]
```

```
                ED              51      FC    A7  F2 00059 4$:    AOBLSS  -4(CETBL), I, 3$                          ; 6661
                                50          6A  9E 0005E         MOVAB   TABLEBASE, R0                             ; 6670
                        040C    C9      04 B640  9E 00061        MOVAB   @4(PTR)[R0], IDENT_OPERATOR_TABLE
                                50          6A  9E 00068         MOVAB   TABLEBASE, R0                             ; 6671
                        0418    C9      08 B640  9E 0006B        MOVAB   @8(PTR)[R0], OPCHAR_OPERATOR_TABLE
                                50          6A  9E 00072         MOVAB   TABLEBASE, R0                             ; 6672
                        0424    C9      0C B640  9E 00075        MOVAB   @12(PTR)[R0], STATE_TABLE
                                50          6A  9E 0007C         MOVAB   TABLEBASE, R0                             ; 6673
                        041C    C9      10 B640  9E 0007F        MOVAB   @16(PTR)[R0], PRIMARY_TABLE
                                50          6A  9E 00086         MOVAB   TABLEBASE, R0                             ; 6674
                        0428    C9      14 B640  9E 00089        MOVAB   @20(PTR)[R0], SUBSCRIPT_TERM_TBL
                                50          6A  9E 00090         MOVAB   TABLEBASE, R0                             ; 6675
                        042C    C9      18 B640  9E 00093        MOVAB   @24(PTR)[R0], PRIDTBL
                                50          6A  9E 0009A         MOVAB   TABLEBASE, R0                             ; 6676
                          F4    A9      1C B640  9E 0009D        MOVAB   @28(PTR)[R0], BIF_TABLE
                        0414    C9      20    A6  D0 000A3       MOVL    32(PTR), MULTIPLE_SUBSCR                  ; 6677
                        0404    C9      24    A6  D0 000A9       MOVL    36(PTR), ENFORCE_RECORD                   ; 6678
                          F8    A9      28    A6  D0 000AF       MOVL    40(PTR), CASING_SIGNIFICANT               ; 6679
                        0400    C9      2C    A6  D0 000B4       MOVL    44(PTR), COMPONENTS_IN_PATHNAME           ; 6680
                        0410    C9      30    A6  D0 000BA       MOVL    48(PTR), INCOMPLETE_QUAL                  ; 6681
                                                  58  DD 000C0   PUSHL   R8                                       ; 6686
                    00000000G   00              01  FB 000C2   CALLS   #1, DBG$EVALOP_SET_LANGUAGE
                                                  58  DD 000C9   PUSHL   R8                                       ; 6687
                    00000000G   00              01  FB 000CB   CALLS   #1, DBG$PRINT_SET_LANGUAGE
                                                  04  000D2   RET                                                 ; 6690

; Routine Size:  211 bytes,    Routine Base:  DBG$CODE + 18BB
```

```
;  6582          6691  1 GLOBAL ROUTINE DBG$PRIMARY_PARSER(OPERAND_EXPECTED_FLAG, ADDRESS_EXPRESSION,
;  6583          6692  1         TERM_LIST, PAREN_NESTING, RET_TOKEN, RET_OPERAND_FLAG): NOVALUE =
;  6584          6693  1 !
;  6585          6694  1 ! FUNCTION
;  6586          6695  1 !       This routine parses Primary Symbols and serves as a get-token routine
;  6587          6696  1 !       for the Expression Parser.  It calls the Lexical Scanner to get lexical
;  6588          6697  1 !       tokens from the command line being parsed.  It then intercepts tokens
;  6589          6698  1 !       which are part of a Primary Symbol and uses those to build a Primary
;  6590          6699  1 !       Descriptor for the symbol.  Lexical tokens which are not part of Prim-
;  6591          6700  1 !       ary Symbols are simply passed through to the Expression Parser.  The
;  6592          6701  1 !       result is that the Expression Parser sees a stream of operators and
;  6593          6702  1 !       operands where each Primary Symbol or constant has been preparsed and
;  6594          6703  1 !       packaged as a single operand by the Primary Parser.
;  6595          6704  1 !
;  6596          6705  1 !       A "Primary Symbol" is defined to be a variable name which may include
;  6597          6706  1 !       pathname qualification, subscripting, data component selection, and
;  6598          6707  1 !       dereferencing.  Exactly which of these are allowed depends on the cur-
;  6599          6708  1 !       rent language.  Thus "X" and "MOD\ROUT\Z" are Primary Symbols and so is
;  6600          6709  1 !       "M\R\X(2,3).Y^.Z(4)".  In effect, a Primary Symbol is anything that can
;  6601          6710  1 !       be described by a Primary Descriptor (see DBGLIB.REQ).
;  6602          6711  1 !
;  6603          6712  1 !       The Primary Parser emulates a Finite-State Machine (FSM) to parse the
;  6604          6713  1 !       Primary Symbols accepted in the current language.  The FMS for the
;  6605          6714  1 !       current language is defined by a Primary Parser State Table which
;  6606          6715  1 !       defines which operators (such as "\", ".", and subscripting) may appear
;  6607          6716  1 !       in which order in a Primary Symbol.  For each transition in the FMS,
;  6608          6717  1 !       a semantic routine is executed which builds up the Primary Descriptor
;  6609          6718  1 !       for the current Primary Symbol (or a Value Descriptor if the current
;  6610          6719  1 !       symbol is a constant).  The symbol is "accepted" by the parser if a
;  6611          6720  1 !       transition is reached which returns the completed Primary Descriptor
;  6612          6721  1 !       to the caller.  If the symbol is not accepted by the FSM, a syntax
;  6613          6722  1 !       error is signalled.
;  6614          6723  1 !
;  6615          6724  1 !       The Primary Parser is called by the Expression Parser.  However, the
;  6616          6725  1 !       Primary Parser will itself call the Expression Parser to pick up sub-
;  6617          6726  1 !       script expressions within Primary Symbols.  Hence these two routines
;  6618          6727  1 !       call each other recursively, and their data structures have been set
;  6619          6728  1 !       up so that this recursion will work properly.
;  6620          6729  1 !
;  6621          6730  1 ! INPUTS
;  6622          6731  1 !       OPERAND_EXPECTED_FLAG - A flag which is set to TRUE if the caller
;  6623          6732  1 !                   expects to see an operand next.  This flag is used to
;  6624          6733  1 !                   determine whether certain operators (such as "+") are
;  6625          6734  1 !                   prefix (if an operand is expected) or infix (if an
;  6626          6735  1 !                   operand is not expected) operators at the current point
;  6627          6736  1 !                   in the parsing of an expression.
;  6628          6737  1 !
;  6629          6738  1 !       ADDRESS_EXPRESSION - A flag set to TRUE if we are parsing a DEBUG
;  6630          6739  1 !                   Address Expression instead of a language expression.  This
;  6631          6740  1 !                   affects the parsing of Address Expression operators such
;  6632          6741  1 !                   as "+", "-", "*", "/", "@", and "^" which are recognized by
;  6633          6742  1 !                   DEBUG rules, not language rules, in Address Expressions.
;  6634          6743  1 !
;  6635          6744  1 !       TERM_LIST - A vector of pointers to Terminator Lexical Token Entries
;  6636          6745  1 !                   for the Terminator Tokens which can terminate the expression
;  6637          6746  1 !                   being parsed.  The vector must be in PLIT form (TERM_LIST[-1]
;  6638          6747  1 !                   gives the number of entries) and each pointer is expected to
```

```
;  6639     6748  1 !                 be relative to TABLEBASE.  If there are no terminator tokens
;  6640     6749  1 !                 other than carriage return, this list is empty (0 entries).
;  6641     6750  1 !
;  6642     6751  1 !       PAREN_NESTING - The current parenthesis nesting depth.  This parameter
;  6643     6752  1 !                 is passed on to the Lexical Scanner which uses it in the
;  6644     6753  1 !                 detection of expression terminator tokens.
;  6645     6754  1 !
;  6646     6755  1 !       RET_TOKEN - The address of a longword to receive a pointer to an Opera-
;  6647     6756  1 !                 tor Lexical Token Entry or a Value or Primary Descriptor.
;  6648     6757  1 !
;  6649     6758  1 !       RET_OPERAND_FLAG - The address of a longword to receive a flag saying
;  6650     6759  1 !                 whether an operator or an operand was returned.
;  6651     6760  1 !
;  6652     6761  1 !       7th parameter - (optional) - if present, indicates partially
;  6653     6762  1 !                 constructed Primary and contains pointer to
;  6654     6763  1 !                 descriptor so far. This is used for example, in the
;  6655     6764  1 !                 C expression (*PTR).COMPONENT, where a Primary is
;  6656     6765  1 !                 returned from the expression parser for (*PTR) and we
;  6657     6766  1 !                 call the Primary Parser to pick up the ".COMPONENT".
;  6658     6767  1 !
;  6659     6768  1 !       8th parameter - (optional) - if present, indicated partially
;  6660     6769  1 !                 constructed Primary and contains starting state.
;  6661     6770  1 !
;  6662     6771  1 ! OUTPUTS
;  6663     6772  1 !       RET_TOKEN - A pointer to an Operator Lexical Token Entry or to a Value
;  6664     6773  1 !                 or Primary Descriptor is returned to RET_TOKEN.  What the
;  6665     6774  1 !                 returned pointer points to is specified by RET_OPERAND_FLAG.
;  6666     6775  1 !
;  6667     6776  1 !       RET_OPERAND_FLAG - A flag value is returned to RET_OPERAND_FLAG.  If
;  6668     6777  1 !                 a pointer to an Operator Lexical Token Entry was returned to
;  6669     6778  1 !                 RET_TOKEN, the value FALSE is returned to RET_OPERAND_FLAG.
;  6670     6779  1 !                 If a pointer to a Primary or Value Descriptor is returned to
;  6671     6780  1 !                 RET_TOKEN, the value TRUE is returned to RET_OPERAND_FLAG.
;  6672     6781  1 !
;  6673     6782  1
;  6674     6783  2     BEGIN
;  6675     6784  2
;  6676     6785  2     MAP
;  6677     6786  2         RET_TOKEN: REF VECTOR[1],       ! Token pointer return location
;  6678     6787  2         RET_OPERAND_FLAG: REF VECTOR[1];! Operand returned flag location
;  6679     6788  2
;  6680     6789  2     BUILTIN
;  6681     6790  2         ACTUALCOUNT,
;  6682     6791  2         ACTUALPARAMETER;
;  6683     6792  2
;  6684     6793  2     OWN
;  6685     6794  2         TOKEN_IS_INTEGER:               ! Lookup table which states whether a
;  6686     6795  2             BITVECTOR[TOKEN$K_MAX_OPERAND + 1]  ! given operand token is some
;  6687     6796  2             PSECT(DBG$PLIT)             ! form of integer or not
;  6688     6797  2             PRESET(
;  6689     6798  2                 [TOKEN$K_INTEGER] = TRUE,
;  6690     6799  2                 [TOKEN$K_HEX_INTEGER] = TRUE,
;  6691     6800  2                 [TOKEN$K_OCT_INTEGER] = TRUE,
;  6692     6801  2                 [TOKEN$K_BIN_INTEGER] = TRUE,
;  6693     6802  2                 [TOKEN$K_PACK_DECIMAL] =TRUE);  !
;  6694     6803  2
;  6695     6804  2     LOCAL
```

```
;6696          6805  2          ACTION,                         ! Current state transition action index
;6697          6806  2          ARG_LIST: REF VECTOR [,LONG],   ! List of built-in function arguments
;6698          6807  2          DUMMY,                          ! Output param for DEFINE lookup - not
;6699          6808  2                                          !     used here.
;6700          6809  2          KIND,                           ! RST symbol kind for current symbol
;6701          6810  2          LAST_OPERAND: REF TOKEN$ENTRY,  ! Pointer to the last operand token
;6702          6811  2                                          !     entry encountered so far
;6703          6812  2          NUMERIC_PATHNAME,               ! Flag set if numeric pathname is used
;6704          6813  2          OPCODE,                         ! Operator code for current operator
;6705          6814  2          OPERAND_EXPECTED,               ! Flag set when operand or prefix ope-
;6706          6815  2                                          !     rator is expected next
;6707          6816  2          PATHDESC: PTH$PATHNAME,         ! Pathname descriptor
;6708          6817  2          PATHSTRING,                     ! Pointer to pathname string for messages
;6709          6818  2          PATHVECTOR: REF VECTOR[,LONG],  ! Pointer to pathname vector in PATHDESC
;6710          6819  2          PLIPTR: REF DBG$PRIMARY,        ! Pointer value
;6711          6820  2          PRID: REF PRID$ENTRY,           ! Pointer to Predefined Identifier Entry
;6712          6821  2          PRIMPTR: REF DBG$PRIMARY,       ! Pointer to Primary Descriptor built
;6713          6822  2          SAVED_PATHDESC: PTH$PATHNAME,   ! Copy of pathname descriptor
;6714          6823  2          STATE_INDEX,                    ! Current index into the Primary Parser
;6715          6824  2                                          !     State Table
;6716          6825  2          STATUS,                         ! Status code returned by RTL routines
;6717          6826  2          STRDESC: BLOCK[8,BYTE],         ! String descriptor for RTL calls
;6718          6827  2          SUBSCR_DESC: SUBSCR$DESC,       ! Holds saved-away subscripts
;6719          6828  2          SYMID,                          ! SYMID (Symbol ID) for current symbol
;6720          6829  2          TEMPTOKEN: REF TOKEN$ENTRY,     ! Pointer to token for invocation number
;6721          6830  2          TOKEN: REF TOKEN$ENTRY,         ! Pointer to the current Token Entry
;6722          6831  2          TYPEID,                         ! TYPEID (Type ID) for current symbol
;6723          6832  2          VALPTR: REF DBG$VALDESC;        ! Pointer to a Value Descriptor
;6724          6833  2
;6725          6834  2
;6726          6835  2
;6727          6836  2      ! There are two different initialization paths. The normal path is
;6728          6837  2      ! when we are picking up a Primary from scratch; that is the ELSE
;6729          6838  2      ! clause below.
;6730          6839  2      !
;6731          6840  2      IF ACTUALCOUNT() GTR 6
;6732          6841  2      THEN
;6733          6842  3          BEGIN
;6734          6843  3
;6735          6844  3
;6736          6845  3          ! If we got an Operator Token last time which was not part of the Primary
;6737          6846  3          ! Symbol we were building, then we saved it in SAVED_TOKEN while we com-
;6738          6847  3          ! pleted and returned the Primary Descriptor.  In that case, return the
;6739          6848  3          ! input Primary and retain the SAVED_TOKEN value for the next time
;6740          6849  3          ! Primary Parser is called.
;6741          6850  3          !
;6742          6851  3          IF .SAVED_TOKEN NEQ 0
;6743          6852  3          THEN
;6744          6853  4              BEGIN
;6745          6854  4              RET_TOKEN[0] = ACTUALPARAMETER(7);
;6746          6855  4              RETURN;
;6747          6856  3              END;
;6748          6857  3
;6749          6858  3
;6750          6859  3          ! This is the case where we call the Primary Parser when we already
;6751          6860  3          ! have constructed part of the Primary, and we want to pick up the
;6752          6861  3          ! rest of the Primary. In this case, a pointer to the partially-
```

```
;6753  6862  3          ! constructed Primary is passed in as the 7th parameter, and the
;6754  6863  3          ! current state in the primary parsing is passed in as the 8th parameter.
;6755  6864  3          ! We also initialize LAST_OPERAND to -1 here; it must be non-zero to
;6756  6865  3          ! avoid confusing the code below.
;6757  6866  3          !
;6758  6867  3          LAST_OPERAND = -1;
;6759  6868  3          OPERAND_EXPECTED = .OPERAND_EXPECTED_FLAG;
;6760  6869  3          STATE_INDEX = ACTUALPARAMETER(8);
;6761  6870  3          PRIMPTR = ACTUALPARAMETER(7);
;6762  6871  3          END
;6763  6872  3
;6764  6873  2      ELSE
;6765  6874  3          BEGIN
;6766  6875  3
;6767  6876  3
;6768  6877  3          ! If we got an Operator Token last time which was not part of the Primary
;6769  6878  3          ! Symbol we were building, then we saved it in SAVED_TOKEN while we com-
;6770  6879  3          ! pleted and returned the Primary Descriptor.  In that case, return the
;6771  6880  3          ! saved token to the Expression Parser now and do no more.
;6772  6881  3          !
;6773  6882  3          IF .SAVED_TOKEN NEQ 0
;6774  6883  3          THEN
;6775  6884  4              BEGIN
;6776  6885  4              TOKEN = .SAVED_TOKEN;
;6777  6886  4              SAVED_TOKEN = 0;
;6778  6887  4              RET_TOKEN[0] = .TOKEN;
;6779  6888  4              RET_OPERAND_FLAG[0] = FALSE;
;6780  6889  4              RETURN;
;6781  6890  4              END;
;6782  6891  3
;6783  6892  3
;6784  6893  3          ! Initialize LAST_OPERAND to be null (no operands have been encountered
;6785  6894  3          ! yet).  Also initialize the OPERAND_EXPECTED flag and various other local
;6786  6895  3          ! variables and data structures.
;6787  6896  3          !
;6788  6897  3          LAST_OPERAND = 0;
;6789  6898  3          OPERAND_EXPECTED = .OPERAND_EXPECTED_FLAG;
;6790  6899  3          STATE_INDEX = 0;
;6791  6900  3          PRIMPTR = 0;
;6792  6901  3          NUMERIC_PATHNAME = FALSE;
;6793  6902  3          CH$FILL(0, DBG$K_PATHNAME_SIZE+%UPVAL, PATHDESC);
;6794  6903  3          PATHVECTOR = PATHDESC[PTH$A_PATHVECTOR];
;6795  6904  3          CH$FILL(0, SUBSCR_DESC_SIZE, SUBSCR_DESC);
;6796  6905  3          PLIPTR = 0;
;6797  6906  3          END;
;6798  6907  2
;6799  6908  2      ! Loop through all lexical tokens on the input line being parsed until we
;6800  6909  2      ! reach a terminator operator.
;6801  6910  2      !
;6802  6911  2      WHILE TRUE DO
;6803  6912  3          BEGIN
;6804  6913  3
;6805  6914  3
;6806  6915  3          ! Get the next lexical token.  DBG$LEXICAL_SCANNER picks up the next
;6807  6916  3          ! token using the rules of the currently set language.
;6808  6917  3          !
;6809  6918  3          TOKEN = DBG$LEXICAL_SCANNER(.OPERAND_EXPECTED,
```

```
;  6810          6919  3                    .ADDRESS_EXPRESSION, .TERM_LIST, .PAREN_NESTING);
;  6811          6920  3            IF .DBG$GL_DEVELOPER[2] THEN DUMP_TOKEN(.TOKEN);
;  6812          6921  3
;  6813          6922  3
;  6814          6923  3            ! Check for an invocation number.  An invocation number consists of an
;  6815          6924  3            ! integer constant in a pathname where an operator is expected.  If
;  6816          6925  3            ! this is an invocation number, we convert it to the invocation number
;  6817          6926  3            ! postfix operator which then passes through the rest of the code below
;  6818          6927  3            ! in the normal way.
;  6819          6928  3            !
;  6820          6929  3            IF (.PRIMPTR EQL 0)                                 AND
;  6821          6930  3               (NOT .OPERAND_EXPECTED)                          AND
;  6822          6931  3               (.LAST_OPERAND NEQ 0)                            AND
;  6823          6932  3               (.TOKEN[TOKEN$B_KIND] EQL TOKEN$K_OPERAND) AND
;  6824          6933  3               .TOKEN_IS_INTEGER[.TOKEN[TOKEN$W_CODE]]
;  6825          6934  3            THEN
;  6826          6935  4                BEGIN
;  6827          6936  4                TEMPTOKEN = DBG$GET_TEMPMEM(TOKEN$K_ENTSIZE_OPERATOR +
;  6828          6937  4                                (.TOKEN[TOKEN$B_LENGTH] + %UPVAL)/%UPVAL);
;  6829          6938  4                TEMPTOKEN[TOKEN$B_KIND] = TOKEN$K_POSTFIX_OP;
;  6830          6939  4                TEMPTOKEN[TOKEN$V_PRIMARY] = TRUE;
;  6831          6940  4                TEMPTOKEN[TOKEN$W_CODE] = TOKEN$K_INVOCNUM;
;  6832          6941  4                CH$MOVE(.TOKEN[TOKEN$B_LENGTH] + 1,
;  6833          6942  4                        TOKEN[TOKEN$B_LENGTH], TEMPTOKEN[TOKEN$B_OPLEN]);
;  6834          6943  4                TOKEN = .TEMPTOKEN;
;  6835          6944  4                END;
;  6836          6945  3
;  6837          6946  3
;  6838          6947  3            ! Handle operands.  If this is an operand, check that we are actually
;  6839          6948  3            ! expecting an operand at this point.  Save a pointer to the operand
;  6840          6949  3            ! and loop to get the next token.
;  6841          6950  3            !
;  6842          6951  3            IF .TOKEN[TOKEN$B_KIND] EQL TOKEN$K_OPERAND
;  6843          6952  3            THEN
;  6844          6953  4                BEGIN
;  6845          6954  4                IF NOT .OPERAND_EXPECTED
;  6846          6955  4                THEN
;  6847          6956  4                    SIGNAL(DBG$_MISINVOPER, 1, TOKEN[TOKEN$B_LENGTH]);
;  6848          6957  4
;  6849          6958  4                OPERAND_EXPECTED = FALSE;
;  6850          6959  4                LAST_OPERAND = .TOKEN;
;  6851          6960  4                END
;  6852          6961  4
;  6853          6962  4
;  6854          6963  4            ! Handle operators.  If this operator is not part of the current Prim-
;  6855          6964  4            ! ary Symbol, we save it while building and returning a descriptor for
;  6856          6965  4            ! the Primary Symbol.  If the operator is part of the current Primary
;  6857          6966  4            ! Symbol, we add to the Primary we are building and loop to pick up
;  6858          6967  4            ! more of the Primary.
;  6859          6968  4            !
;  6860          6969  3            ELSE
;  6861          6970  4                BEGIN
;  6862          6971  4
;  6863          6972  4
;  6864          6973  4                ! If this operator is not part of the Primary Symbol we are build-
;  6865          6974  4                ! ing (if any), then it is a language or address expression opera-
;  6866          6975  4                ! tor.  Save it for the next call on DBG$PRIMARY_PARSER and use the
```

```
; 6867          6976  4          ! Terminator Token to close out the Primary Symbol we are building
; 6868          6977  4          ! (if we are building one).
; 6869          6978  4          !
; 6870          6979  4          IF NOT .TOKEN[TOKEN$V_PRIMARY]
; 6871          6980  4          THEN
; 6872          6981  5              BEGIN
; 6873          6982  5
; 6874          6983  5
; 6875          6984  5              ! If no operand was started, return the operator immediately.
; 6876          6985  5              !
; 6877          6986  5              IF .LAST_OPERAND EQL 0
; 6878          6987  5              THEN
; 6879          6988  6                  BEGIN
; 6880          6989  6                  RET_TOKEN[0] = .TOKEN;
; 6881          6990  6                  RET_OPERAND_FLAG[0] = FALSE;
; 6882          6991  6                  RETURN;
; 6883          6992  5                  END;
; 6884          6993  5
; 6885          6994  5
; 6886          6995  5              ! An operand is present--save the language operator and set
; 6887          6996  5              ! things up to close out and return the operand.
; 6888          6997  5              !
; 6889          6998  5              SAVED_TOKEN = .TOKEN;
; 6890          6999  5              TOKEN = PRIMARY_TERM_TOKEN;
; 6891          7000  5              RET_OPERAND_FLAG[0] = TRUE;
; 6892          7001  4              END;
; 6893          7002  4
; 6894          7003  4
; 6895          7004  4          ! We now have a Primary operator.  Check that an operator was
; 6896          7005  4          ! expected unless this is a prefix operator (which is okay when
; 6897          7006  4          ! we expect an operand).  This check catches many kinds of ill-
; 6898          7007  4          ! formed Primary Symbols.  Also say that we expect an operand
; 6899          7008  4          ! next unless this is a postfix operator.
; 6900          7009  4          !
; 6901          7010  5          IF (.OPERAND_EXPECTED AND
; 6902          7011  4                  (.TOKEN[TOKEN$B_KIND] NEQ TOKEN$K_PREFIX_OP)) OR
; 6903          7012  5             ((NOT .OPERAND_EXPECTED) AND
; 6904          7013  5                  (.TOKEN[TOKEN$B_KIND] EQL TOKEN$K_PREFIX_OP))
; 6905          7014  4          THEN
; 6906          7015  4              SIGNAL(DBG$_MISOPEMIS, 1, TOKEN[TOKEN$B_OPLEN]);
; 6907          7016  4
; 6908          7017  4
; 6909          7018  4          IF .TOKEN[TOKEN$B_KIND] NEQ TOKEN$K_PREFIX_OP
; 6910          7019  4          THEN
; 6911          7020  4              IF .LAST_OPERAND EQL 0
; 6912          7021  4              THEN
; 6913          7022  4                  SIGNAL(DBG$_MISOPEMIS, 1, TOKEN[TOKEN$B_OPLEN])
; 6914          7023  4              ELSE
; 6915          7024  5                  BEGIN
; 6916          7025  5                  IF .LAST_OPERAND NEQ -1
; 6917          7026  5                  THEN
; 6918          7027  6                      BEGIN
; 6919          7028  7                      IF (.LAST_OPERAND[TOKEN$B_KIND] NEQ TOKEN$K_IDENTIFIER)
; 6920          7029  6                      THEN
; 6921          7030  6                          SIGNAL(DBG$_MISOPEMIS, 1, TOKEN[TOKEN$B_OPLEN]);
; 6922          7031  5                      END;
; 6923          7032  4                  END;
```

```
;  6924          7033  4
;  6925          7034  4
;  6926          7035  4          IF .TOKEN[TOKEN$B_KIND] NEQ TOKEN$K_POSTFIX_OP
;  6927          7036  4          THEN
;  6928          7037  4              OPERAND_EXPECTED = TRUE;
;  6929          7038  4
;  6930          7039  4
;  6931          7040  4          ! Get the Operator Code for this Primary Operator and loop through
;  6932          7041  4          ! the transitions for the current state in the Primary Parser State
;  6933          7042  4          ! Table until we find a transition for this operator.  If we find
;  6934          7043  4          ! no such transition (PRIMARY$B_OPCODE field zero), the current
;  6935          7044  4          ! operator is not allowed in this context, so we signal a syntax
;  6936          7045  4          ! error.  If the transition is allowed, we pick up its action index
;  6937          7046  4          ! and the next state in the FSM.
;  6938          7047  4          !
;  6939          7048  4          OPCODE = .TOKEN[TOKEN$W_CODE];
;  6940          7049  4          WHILE .PRIMARY_TABLE[.STATE_INDEX, PRIMARY$B_OPCODE] NEQ .OPCODE DO
;  6941          7050  5              BEGIN
;  6942          7051  5              IF .PRIMARY_TABLE[.STATE_INDEX, PRIMARY$B_OPCODE] EQL 0
;  6943          7052  5              THEN
;  6944          7053  5                  SIGNAL(DBG$_SYNERREXPR, 1, TOKEN[TOKEN$B_OPLEN]);
;  6945          7054  5
;  6946          7055  5              STATE_INDEX = .STATE_INDEX + 1;
;  6947          7056  4              END;
;  6948          7057  4
;  6949          7058  4          ACTION = .PRIMARY_TABLE[.STATE_INDEX, PRIMARY$B_ACTION];
;  6950          7059  4          STATE_INDEX = .PRIMARY_TABLE[.STATE_INDEX, PRIMARY$W_NEXTSTATE];
;  6951          7060  4
;  6952          7061  4
;  6953          7062  4          ! Execute the action routine associated with this state transition.
;  6954          7063  4          !
;  6955          7064  4          CASE .ACTION FROM PRIMARY$K_MIN_ACTION TO PRIMARY$K_MAX_ACTION OF
;  6956          7065  4              SET
;  6957          7066  4
;  6958          7067  4
;  6959          7068  4              ! Handle Global Symbol backslash operator (prefix "\").  Do
;  6960          7069  4              ! nothing at this point.
;  6961          7070  4              !
;  6962          7071  4              [PRIMARY$K_ACT_START_GBL]:
;  6963          7072  4                  0;
;  6964          7073  4
;  6965          7074  4
;  6966          7075  4              ! Handle terminator after Global Symbol backslash.  Just pick
;  6967          7076  4              ! up the global symbol name and create a Primary Descriptor
;  6968          7077  4              ! for it.  Then exit from the parse loop.
;  6969          7078  4              !
;  6970          7079  4              [PRIMARY$K_ACT_GBL_TERM]:
;  6971          7080  5                  BEGIN
;  6972          7081  5                  PATHDESC[PTH$B_TOTCNT] = 1;
;  6973          7082  5                  PATHDESC[PTH$B_PATHCNT] = 1;
;  6974          7083  5                  PATHVECTOR[0] = UPLIT BYTE(0);
;  6975          7084  5                  APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, NOT_REC_COMP);
;  6976          7085  5                  PRIMPTR = PATHNAME_TO_PRIMARY(PATHDESC, SUBSCR_DESC,
;  6977          7086  5                                                .PLIPTR, SAVED_PATHDESC);
;  6978          7087  5                  EXITLOOP;
;  6979          7088  4                  END;
;  6980          7089  4
```

```
;  6981          7090  4
;  6982          7091  4                ! Handle backslash immediately after the start of the symbol.
;  6983          7092  4                ! If the last operand was a number, we have a numeric pathname
;  6984          7093  4                ! and we put that into the Pathname Descriptor and set the
;  6985          7094  4                ! numeric pathname flag.  If it is an identifier, we simply
;  6986          7095  4                ! append it to the Pathname Descriptor as is.
;  6987          7096  4                !
;  6988          7097  4                [PRIMARY$K_ACT_START_SLASH]:
;  6989          7098  5                    BEGIN
;  6990          7099  5
;  6991          7100  5
;  6992          7101  5                    ! If the last operand is an integer, then it constitutes
;  6993          7102  5                    ! a numeric pathname.  We put that in the Pathname Descrip-
;  6994          7103  5                    ! tor and set the NUMERIC_PATHNAME flag (which means that
;  6995          7104  5                    ! no additional pathname qualification is allowed).
;  6996          7105  5                    !
;  6997          7106  5                    IF .TOKEN_IS_INTEGER[.LAST_OPERAND[TOKEN$W_CODE]]
;  6998          7107  5                    THEN
;  6999          7108  6                        BEGIN
;  7000          7109  6                        STRDESC[DSC$B_DTYPE] = DSC$K_DTYPE_T;
;  7001          7110  6                        STRDESC[DSC$B_CLASS] = DSC$K_CLASS_S;
;  7002          7111  6                        STRDESC[DSC$W_LENGTH] = .LAST_OPERAND[TOKEN$B_LENGTH];
;  7003          7112  6                        STRDESC[DSC$A_POINTER] = LAST_OPERAND[TOKEN$A_NAME];
;  7004          7113  6                        STATUS = OTS$CVT_TI_L(STRDESC, PATHDESC[PTH$L_INVOCNUM]);
;  7005          7114  6                        IF NOT .STATUS
;  7006          7115  6                        THEN
;  7007          7116  6                            SIGNAL(DBG$_ILLPATHELEM, 1,
;  7008          7117  6                                LAST_OPERAND[TOKEN$B_LENGTH], .STATUS);
;  7009          7118  6
;  7010          7119  6                        PATHDESC[PTH$B_LOCINVOC] = 1;
;  7011          7120  6                        PATHDESC[PTH$B_TOTCNT] = 1;
;  7012          7121  6                        PATHDESC[PTH$B_PATHCNT] = 1;
;  7013          7122  6                        PATHVECTOR[0] = UPLIT BYTE(0);
;  7014          7123  6                        NUMERIC_PATHNAME = TRUE;
;  7015          7124  6                        END
;  7016          7125  6
;  7017          7126  6
;  7018          7127  6
;  7019          7128  6                    ! Otherwise simply append the last operand to the current
;  7020          7129  6                    ! Pathname Descriptor.  (The append routine checks that
;  7021          7130  6                    ! the operand is an identifier.)
;  7022          7131  6                    !
;  7023          7132  5                    ELSE
;  7024          7133  5                        APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, NOT_REC_COMP);
;  7025          7134  5
;  7026          7135  4                    END;
;  7027          7136  4
;  7028          7137  4
;  7029          7138  4                ! Handle an invocation number immediately after the first or a
;  7030          7139  4                ! later pathname component of the symbol.  Check the invocation
;  7031          7140  4                ! number for validity and fill it into the Pathname Descriptor.
;  7032          7141  4                !
;  7033          7142  4                [PRIMARY$K_ACT_SLASH_INVOCNUM]:
;  7034          7143  5                    BEGIN
;  7035          7144  5
;  7036          7145  5
;  7037          7146  5                    ! Check that an invocation number has not already been
```

```
;  7038     7147  5                ! specified--if one has, signal an error.
;  7039     7148  5                !
;  7040     7149  5                IF .PATHDESC[PTH$B_LOCINVOC] NEQ 0
;  7041     7150  5                THEN
;  7042     7151  5                    SIGNAL(DBG$_TOOMANINV);
;  7043     7152  5
;  7044     7153  5
;  7045     7154  5                ! Convert the invocation number to internal form and fill
;  7046     7155  5                ! it into the Pathname Descriptor.
;  7047     7156  5                !
;  7048     7157  5                STRDESC[DSC$B_DTYPE] = DSC$K_DTYPE_T;
;  7049     7158  5                STRDESC[DSC$B_CLASS] = DSC$K_CLASS_S;
;  7050     7159  5                STRDESC[DSC$W_LENGTH] = .TOKEN[TOKEN$B_OPLEN];
;  7051     7160  5                STRDESC[DSC$A_POINTER] = TOKEN[TOKEN$A_OPNAME];
;  7052     7161  5                STATUS = OTS$CVT_TI_L(STRDESC, PATHDESC[PTH$L_INVOCNUM]);
;  7053     7162  5                IF NOT .STATUS
;  7054     7163  5                THEN
;  7055     7164  5                    SIGNAL(DBG$_ILLPATHELEM, 1, TOKEN[TOKEN$B_OPLEN],
;  7056     7165  5                            DBG$_UNACVT, 3, UPLIT BYTE(%ASCIC 'decimal '),
;  7057     7166  5                            .STRDESC, UPLIT BYTE(%ASCIC 'longword integer'),
;  7058     7167  5                            .STATUS);
;  7059     7168  5
;  7060     7169  5                PATHDESC[PTH$B_LOCINVOC] = .PATHDESC[PTH$B_TOTCNT] + 1;
;  7061     7170  4                END;
;  7062     7171  4
;  7063     7172  4
;  7064     7173  4            ! Handle dot immediately after the start of the symbol.  Append
;  7065     7174  4            ! the last operand to the current Pathname Descriptor and build
;  7066     7175  4            ! the first part of a Primary Descriptor for it.
;  7067     7176  4            !
;  7068     7177  4            [PRIMARY$K_ACT_START_DOT]:
;  7069     7178  5                BEGIN
;  7070     7179  5                LABEL TEMP_BLOCK;
;  7071     7180  5
;  7072     7181  5                ! If the last operand was an identifier, we append it to
;  7073     7182  5                ! the current Pathname Descriptor and convert that to a
;  7074     7183  5                ! Primary Descriptor.
;  7075     7184  5                !
;  7076     7185  6                TEMP_BLOCK: BEGIN
;  7077     7186  6                IF .LAST_OPERAND[TOKEN$W_CODE] EQL TOKEN$K_IDENTIFIER
;  7078     7187  6                THEN
;  7079     7188  7                    BEGIN
;  7080     7189  7
;  7081     7190  7
;  7082     7191  7                    ! First check for DEFINEd symbols.
;  7083     7192  7                    ! Check that no invocation number is present.
;  7084     7193  7                    !
;  7085     7194  7                    IF .PATHDESC[PTH$B_LOCINVOC] EQL 0
;  7086     7195  7                    THEN
;  7087     7196  7
;  7088     7197  7
;  7089     7198  7                        ! Look up the symbol in the DEFINE symbol table.
;  7090     7199  7                        !
;  7091     7200  8                        BEGIN
;  7092     7201  8                        IF DBG$DEF_SYM_FIND (LAST_OPERAND [TOKEN$B_LENGTH],
;  7093     7202  8                                             KIND, PRIMPTR,
;  7094     7203  8                                             DUMMY, DUMMY)
```

```
;  7095          7204  8                    THEN
;  7096          7205  9                        BEGIN
;  7097          7206  9                        IF .KIND EQL DEFINE_ADDRESS
;  7098          7207  9                        OR .KIND EQL DEFINE_VALUE
;  7099          7208  9                        THEN
;  7100          7209 10                            BEGIN
;  7101          7210 10
;  7102          7211 10
;  7103          7212 10                            ! We have found a matching DEFINEd symbol.
;  7104          7213 10                            ! Copy the descriptor into temporary memory.
;  7105          7214 10                            ! (Fourth parameter FALSE <-> copy into tempmem).
;  7106          7215 10                            !
;  7107          7216 10                            DBG$NCOPY_DESC (.PRIMPTR, PRIMPTR,
;  7108          7217 10                                            DUMMY, FALSE);
;  7109          7218 10                            LEAVE TEMP_BLOCK;
;  7110          7219  9                            END;
;  7111          7220  8                        END;
;  7112          7221  7                    END;
;  7113          7222  6                END;
;  7114          7223  6                APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, NOT_REC_COMP);
;  7115          7224  6                PRIMPTR = PATHNAME_TO_PRIMARY(PATHDESC,SUBSCR_DESC,
;  7116          7225  6                                              .PLIPTR,SAVED_PATHDESC);
;  7117          7226  5                END; ! TEMP_BLOCK
;  7118          7227  5
;  7119          7228  4                END;
;  7120          7229  4
;  7121          7230  4
;  7122          7231  4            ! Handle a dot immediately after the start of the symbol.
;  7123          7232  4            ! In PLI we collect all the record components before
;  7124          7233  4            ! calling PATHNAME_TO_PRIMARY.
;  7125          7234  4            !
;  7126          7235  4            [PRIMARY$K_ACT_START_DOT_PLI]:
;  7127          7236  5                BEGIN
;  7128          7237  5                APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, NOT_REC_COMP);
;  7129          7238  4                END;
;  7130          7239  4
;  7131          7240  4
;  7132          7241  4            ! Handle a dot after the start of the symbol in COBOL.
;  7133          7242  4            !
;  7134          7243  4            [PRIMARY$K_ACT_START_DOT_COB]:
;  7135          7244  5                BEGIN
;  7136          7245  5                APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, COB_REC_COMP);
;  7137          7246  4                END;
;  7138          7247  4
;  7139          7248  4
;  7140          7249  4            ! Handle a subscript parenthesis immediately after the start of
;  7141          7250  4            ! the symbol.  Append the last operand to the Pathname Descrip-
;  7142          7251  4            ! tor, build a partial Primary Descriptor, and then pick up the
;  7143          7252  4            ! subscript expressions within the parentheses.
;  7144          7253  4            !
;  7145          7254  4            [PRIMARY$K_ACT_START_SUBSCR]:
;  7146          7255  5                BEGIN
;  7147          7256  5                LABEL TEMP_BLOCK;
;  7148          7257  5
;  7149          7258  5                ! If the last operand was an identifier, we append it to
;  7150          7259  5                ! the current Pathname Descriptor and convert that to a
;  7151          7260  5                ! Primary Descriptor.
```

```
;  7152     7261  5                !
;  7153     7262  6                TEMP_BLOCK: BEGIN
;  7154     7263  6                IF .LAST_OPERAND[TOKEN$W_CODE] EQL TOKEN$K_IDENTIFIER
;  7155     7264  6                THEN
;  7156     7265  7                    BEGIN
;  7157     7266  7
;  7158     7267  7                    ! First check for DEFINEd symbols.
;  7159     7268  7                    ! Check that no invocation number is present.
;  7160     7269  7                    !
;  7161     7270  7                    IF .PATHDESC[PTH$B_LOCINVOC] EQL 0
;  7162     7271  7                    THEN
;  7163     7272  7
;  7164     7273  7                        ! Look up the symbol in the DEFINE symbol table.
;  7165     7274  7                        !
;  7166     7275  7                        BEGIN
;  7167     7276  7                        IF DBG$DEF_SYM_FIND (LAST_OPERAND [TOKEN$B_LENGTH],
;  7168     7277  8                                             KIND, PRIMPTR,
;  7169     7278  8                                             DUMMY, DUMMY)
;  7170     7279  8                        THEN
;  7171     7280  8                            BEGIN
;  7172     7281  8                            IF .KIND EQL DEFINE_ADDRESS
;  7173     7282  9                            OR .KIND EQL DEFINE_VALUE
;  7174     7283  9                            THEN
;  7175     7284  9                                BEGIN
;  7176     7285  9
;  7177     7286 10                                ! We have found a matching DEFINEd symbol.
;  7178     7287 10                                ! Copy the descriptor into temporary memory.
;  7179     7288 10                                ! (Fourth parameter FALSE <-> copy into tempmem).
;  7180     7289 10                                !
;  7181     7290 10                                DBG$NCOPY_DESC (.PRIMPTR, PRIMPTR,
;  7182     7291 10                                                DUMMY, FALSE);
;  7183     7292 10                                LEAVE TEMP_BLOCK;
;  7184     7293 10                                END;
;  7185     7294 10                            END;
;  7186     7295 10                        END;
;  7187     7296  9                    END;
;  7188     7297  8                APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, NOT_REC_COMP);
;  7189     7298  7                PRIMPTR = PATHNAME_TO_PRIMARY(PATHDESC,SUBSCR_DESC,
;  7190     7299  6                                              .PLIPTR, SAVED_PATHDESC, TRUE);
;  7191     7300  6                END; ! TEMP_BLOCK
;  7192     7301  6
;  7193     7302  6                GET_SUBSCRIPTS(.PRIMPTR);
;  7194     7303  5                END;
;  7195     7304  5
;  7196     7305  5            ! Handle a subscript immediately after the start of the
;  7197     7306  4            ! symbol. In PLI we save away the subscripts and do not
;  7198     7307  4            ! build them into the Primary until later.
;  7199     7308  4            !
;  7200     7309  4            [PRIMARY$K_ACT_START_SUBSCR_PLI]:
;  7201     7310  4                BEGIN
;  7202     7311  4                APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, NOT_REC_COMP);
;  7203     7312  4                SAVE_SUBSCRIPTS(PATHDESC, SUBSCR_DESC);
;  7204     7313  4                END;
;  7205     7314  5
;  7206     7315  5
;  7207     7316  5
;  7208     7317  4
```

```
;  7209     7318  4
;  7210     7319  4
;  7211     7320  4          ! Handle a subscript parenthesis immediately after the start of
;  7212     7321  4          ! the symbol.  Append the last operand to the Pathname Descrip-
;  7213     7322  4          ! tor, build a partial Primary Descriptor, and then pick up the
;  7214     7323  4          ! subscript expressions within the square brackets. BLISS needs to
;  7215     7324  4          ! be special cased because BLISS structures are represented
;  7216     7325  4          ! differently in the RST and DST.
;  7217     7326  4          !
;  7218     7327  4          [PRIMARY$K_ACT_START_SUBSCR_BLI]:
;  7219     7328  5              BEGIN
;  7220     7329  5              LABEL TEMP_BLOCK;
;  7221     7330  5
;  7222     7331  5              ! If the last operand was an identifier, we
;  7223     7332  6              TEMP_BLOCK: BEGIN
;  7224     7333  6              IF .LAST_OPERAND[TOKEN$W_CODE] EQL TOKEN$K_IDENTIFIER
;  7225     7334  6              THEN
;  7226     7335  7                  BEGIN
;  7227     7336  7
;  7228     7337  7
;  7229     7338  7                  ! First check for DEFINEd symbols.
;  7230     7339  7                  ! Check that no invocation number is present.
;  7231     7340  7                  !
;  7232     7341  7                  IF .PATHDESC[PTH$B_LOCINVOC] EQL 0
;  7233     7342  7                  THEN
;  7234     7343  7
;  7235     7344  7
;  7236     7345  7                      ! Look up the symbol in the DEFINE symbol table.
;  7237     7346  7                      !
;  7238     7347  8                      BEGIN
;  7239     7348  8                      IF DBG$DEF_SYM_FIND (LAST_OPERAND [TOKEN$B_LENGTH],
;  7240     7349  8                                           KIND, PRIMPTR,
;  7241     7350  8                                           DUMMY, DUMMY)
;  7242     7351  8                      THEN
;  7243     7352  9                          BEGIN
;  7244     7353  9                          IF .KIND EQL DEFINE_ADDRESS
;  7245     7354  9                          OR .KIND EQL DEFINE_VALUE
;  7246     7355  9                          THEN
;  7247     7356 10                              BEGIN
;  7248     7357 10
;  7249     7358 10
;  7250     7359 10                              ! We have found a matching DEFINEd symbol.
;  7251     7360 10                              ! Copy the descriptor into temporary memory.
;  7252     7361 10                              ! (Fourth parameter FALSE <-> copy into tempmem).
;  7253     7362 10                              !
;  7254     7363 10                              DBG$NCOPY_DESC (.PRIMPTR, PRIMPTR,
;  7255     7364 10                                              DUMMY, FALSE);
;  7256     7365 10                              LEAVE TEMP_BLOCK;
;  7257     7366  9                              END;
;  7258     7367  8                          END;
;  7259     7368  7                      END;
;  7260     7369  7              END
;  7261     7370  7
;  7262     7371  6          ELSE
;  7263     7372  6
;  7264     7373  6              ! If the last operand was not an identifier then
;  7265     7374  6              ! it is not legal to select a component.
```

```
;  7266     7375  6                    !
;  7267     7376  6                    SIGNAL(DBG$_NOTASTRUCT, 1, LAST_OPERAND[TOKEN$B_LENGTH]);
;  7268     7377  6
;  7269     7378  6                ! Append the last operand to
;  7270     7379  6                ! the current Pathname Descriptor and convert that to a
;  7271     7380  6                ! Primary Descriptor.
;  7272     7381  6                !
;  7273     7382  6                APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, NOT_REC_COMP);
;  7274     7383  6                PRIMPTR = PATHNAME_TO_PRIMARY(PATHDESC,SUBSCR_DESC,
;  7275     7384  6                                              .PLIPTR,SAVED_PATHDESC);
;  7276     7385  5                END; ! TEMP_BLOCK
;  7277     7386  5
;  7278     7387  5                GET_BLISS_SUBSCRIPTS(.PRIMPTR, LAST_OPERAND[TOKEN$B_LENGTH]);
;  7279     7388  4                END;
;  7280     7389  4
;  7281     7390  4
;  7282     7391  4            ! Handler the PASCAL dereference operator (^).  Append the
;  7283     7392  4            ! last identifier onto the pathname.  Then build the pathname
;  7284     7393  4            ! into a Primary Descriptor.  Then call the routine that
;  7285     7394  4            ! handles derefencing --it just lights the EVAL bit on the
;  7286     7395  4            ! sub-node and then allocates a new subnode for the object
;  7287     7396  4            ! being pointed to.
;  7288     7397  4            !
;  7289     7398  4            [PRIMARY$K_ACT_START_DEREF]:
;  7290     7399  5                BEGIN
;  7291     7400  5                LABEL TEMP_BLOCK;
;  7292     7401  5
;  7293     7402  5                ! If the last operand was an identifier, we append it to
;  7294     7403  5                ! the current Pathname Descriptor and convert that to a
;  7295     7404  5                ! Primary Descriptor.
;  7296     7405  5                !
;  7297     7406  6                TEMP_BLOCK: BEGIN
;  7298     7407  6                IF .LAST_OPERAND[TOKEN$W_CODE] EQL TOKEN$K_IDENTIFIER
;  7299     7408  6                THEN
;  7300     7409  7                    BEGIN
;  7301     7410  7
;  7302     7411  7
;  7303     7412  7                    ! First check for DEFINEd symbols.
;  7304     7413  7                    ! Check that no invocation number is present.
;  7305     7414  7                    !
;  7306     7415  7                    IF .PATHDESC[PTH$B_LOCINVOC] EQL 0
;  7307     7416  7                    THEN
;  7308     7417  7
;  7309     7418  7
;  7310     7419  7                        ! Look up the symbol in the DEFINE symbol table.
;  7311     7420  7                        !
;  7312     7421  8                        BEGIN
;  7313     7422  8                        IF DBG$DEF_SYM_FIND (LAST_OPERAND [TOKEN$B_LENGTH],
;  7314     7423  8                                             KIND, PRIMPTR,
;  7315     7424  8                                             DUMMY, DUMMY)
;  7316     7425  8                        THEN
;  7317     7426  9                            BEGIN
;  7318     7427  9                            IF .KIND EQL DEFINE_ADDRESS
;  7319     7428  9                            OR .KIND EQL DEFINE_VALUE
;  7320     7429  9                            THEN
;  7321     7430 10                                BEGIN
;  7322     7431 10
```

```
;  7323          7432  10                        ! We have found a matching DEFINEd symbol.
;  7324          7433  10                        ! Copy the descriptor into temporary memory.
;  7325          7434  10                        ! (Fourth parameter FALSE <-> copy into tempmem).
;  7326          7435  10                        !
;  7327          7436  10                        DBG$NCOPY_DESC (.PRIMPTR, PRIMPTR,
;  7328          7437  10                                        DUMMY, FALSE);
;  7329          7438  10                        LEAVE TEMP_BLOCK;
;  7330          7439  10                        END;
;  7331          7440   9                    END;
;  7332          7441   8                END;
;  7333          7442   7            END;
;  7334          7443   6        END;
;  7335          7444   6    APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, NOT_REC_COMP);
;  7336          7445   6    PRIMPTR = PATHNAME_TO_PRIMARY(PATHDESC,SUBSCR_DESC,
;  7337          7446   6                                  .PLIPTR,SAVED_PATHDESC);
;  7338          7447   5    END; ! TEMP_BLOCK
;  7339          7448   5
;  7340          7449   5    GET_DEREFERENCE(.PRIMPTR);
;  7341          7450   4    END;
;  7342          7451   4
;  7343          7452   4
;  7344          7453   4    ! Handle a PLI dereference as in A->B. Here we convert the
;  7345          7454   4    ! left-hand-side to a Primary, and then save away the Primary.
;  7346          7455   4    ! The parse is then "backed up" to the start state to pick
;  7347          7456   4    ! up the right hand side of the "->". Later, in the
;  7348          7457   4    ! PATHNAME_TO_PRIMARY routine, the two primaries are glued
;  7349          7458   4    ! together.
;  7350          7459   4    !
;  7351          7460   4    [PRIMARY$K_ACT_START_DEREF_PLI]:
;  7352          7461   5        BEGIN
;  7353          7462   5
;  7354          7463   5        ! Save away the Primary for the stuff to the left of "->".
;  7355          7464   5        !
;  7356          7465   5        APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, NOT_REC_COMP);
;  7357          7466   5        PLIPTR = PATHNAME_TO_PRIMARY(PATHDESC,SUBSCR_DESC,
;  7358          7467   5                                     .PLIPTR,SAVED_PATHDESC);
;  7359          7468   5        IF (.PLIPTR[DBG$B_DHDR_KIND] NEQ RST$K_DATA) OR
;  7360          7469   6           (.PLIPTR[DBG$B_DHDR_FCODE] NEQ RST$R_TYPE_PTR)
;  7361          7470   5        THEN
;  7362          7471   6            BEGIN
;  7363          7472   6            LOCAL
;  7364          7473   6                NAME;
;  7365          7474   6            DBG$NPATHDESC_TO_CS(PATHDESC, NAME);
;  7366          7475   6            SIGNAL(DBG$_VALNOTADDR, 1, .NAME);
;  7367          7476   5            END;
;  7368          7477   5
;  7369          7478   5        ! Re-initialize.
;  7370          7479   5        !
;  7371          7480   5        PRIMPTR = 0;
;  7372          7481   5        NUMERIC_PATHNAME = FALSE;
;  7373          7482   5        CH$FILL(0, DBG$K_PATHNAME_SIZE*%UPVAL, PATHDESC);
;  7374          7483   5        PATHVECTOR = PATHDESC[PTH$A_PATHVECTOR];
;  7375          7484   5        CH$FILL(0, SUBSCR_DESC_SIZE, SUBSCR_DESC);
;  7376          7485   4        END;
;  7377          7486   4
;  7378          7487   4
;  7379          7488   4    ! Handle the built-in function operator that follows the
```

```
;  7380     7489  4                ! built-in function name.  This will return a primary that
;  7381     7490  4                ! represents the value of the built-in function.
;  7382     7491  4                !
;  7383     7492  4                [PRIMARY$K_ACT_START_BIF_CALL]:
;  7384     7493  5                    BEGIN
;  7385     7494  5
;  7386     7495  5
;  7387     7496  5                    ! If the last operand was not a built-in function name,
;  7388     7497  5                    ! signal the error.
;  7389     7498  5                    !
;  7390     7499  5                    IF .LAST_OPERAND[TOKEN$W_CODE] NEQ TOKEN$K_BUILTIN_FUNCTION
;  7391     7500  5                    THEN
;  7392     7501  5                        SIGNAL(DBG$_MISOPEMIS, 1, TOKEN[TOKEN$B_OPLEN]);
;  7393     7502  5
;  7394     7503  5                    ! Call a routine that returns a counted longword vector of
;  7395     7504  5                    ! pointers that point to primary tokens of the arguments
;  7396     7505  5                    ! of the specified built-in function call.  The number of
;  7397     7506  5                    ! arguments expected in the B_FLAGS field of a built-in
;  7398     7507  5                    ! function operand.
;  7399     7508  5                    !
;  7400     7509  5                    ARG_LIST = DBG$GET_BIF_ARGUMENTS(.LAST_OPERAND[TOKEN$B_FLAGS],
;  7401     7510  5                                                     LAST_OPERAND[TOKEN$B_LENGTH]);
;  7402     7511  5
;  7403     7512  5                    ! For now, we will not accept any built-in function that
;  7404     7513  5                    ! contains more than 2 arguments.  If more than 2 are
;  7405     7514  5                    ! present, an invalid argument list message in signaled.
;  7406     7515  5                    !
;  7407     7516  5                    IF .ARG_LIST[0] LEQ 2
;  7408     7517  5                    THEN
;  7409     7518  6                        BEGIN
;  7410     7519  6
;  7411     7520  6                        ! Create a new operator token to represent the specific
;  7412     7521  6                        ! built-in function call and then pass this token to
;  7413     7522  6                        ! DBG$EVAL_LANG_OPERATOR to proccess the built-in
;  7414     7523  6                        ! function and return its value as a primary.
;  7415     7524  6                        !
;  7416     7525  6                        IF .ARG_LIST[0] EQL 1
;  7417     7526  6                        THEN
;  7418     7527  6                            TOKEN = CREATE_OPERATOR_TOKEN(.LAST_OPERAND[TOKEN$B_BIF],
;  7419     7528  6                                                          LAST_OPERAND[TOKEN$B_LENGTH],
;  7420     7529  6                                                          .TOKEN[TOKEN$B_KIND])
;  7421     7530  6                        ELSE
;  7422     7531  6                            TOKEN = CREATE_OPERATOR_TOKEN(.LAST_OPERAND[TOKEN$B_BIF],
;  7423     7532  6                                                          LAST_OPERAND[TOKEN$B_LENGTH],
;  7424     7533  6                                                          TOKEN$K_INFIX_OP);
;  7425     7534  6
;  7426     7535  6                        PRIMPTR = DBG$EVAL_LANG_OPERATOR(.TOKEN, .ARG_LIST[1], .ARG_LIST[2]);
;  7427     7536  6
;  7428     7537  6                        ! Return the fact that we have built the result by now
;  7429     7538  6                        ! and that it is a value descriptor.  Because of the
;  7430     7539  6                        ! way the BIF tokens come through the above initialization
;  7431     7540  6                        ! code, this value was never set.  And It Better Be!
;  7432     7541  6                        !
;  7433     7542  6                        RET_OPERAND_FLAG[0] = TRUE;
;  7434     7543  6
;  7435     7544  6                        END
;  7436     7545  5                    ELSE
```

```
;  7437     7546  5                    SIGNAL(DBG$_INVARGLIS, 1, LAST_OPERAND[TOKEN$B_LENGTH]);
;  7438     7547  5                EXITLOOP;
;  7439     7548  4                END;
;  7440     7549  4
;  7441     7550  4
;  7442     7551  4            ! Handle the Ada tick operator immediately after the start of
;  7443     7552  4            ! the symbol.  The symbol thus consists of just a single type
;  7444     7553  4            ! name.
;  7445     7554  4            !
;  7446     7555  4            [PRIMARY$K_ACT_START_TICK]:
;  7447     7556  5                BEGIN
;  7448     7557  5
;  7449     7558  5
;  7450     7559  5                ! If the last operand was an identifier, we append it to
;  7451     7560  5                ! the current Pathname Descriptor and convert that to a
;  7452     7561  5                ! Primary Descriptor.
;  7453     7562  5                !
;  7454     7563  5                IF .LAST_OPERAND[TOKEN$W_CODE] EQL TOKEN$K_IDENTIFIER
;  7455     7564  5                THEN
;  7456     7565  6                    BEGIN
;  7457     7566  6
;  7458     7567  6
;  7459     7568  6                    ! First check for DEFINEd symbols.
;  7460     7569  6                    ! Check that no invocation number is present.
;  7461     7570  6                    !
;  7462     7571  6                    IF .PATHDESC[PTH$B_LOCINVOC] EQL 0
;  7463     7572  6                    THEN
;  7464     7573  6
;  7465     7574  6
;  7466     7575  6                        ! Look up the symbol in the DEFINE symbol table.
;  7467     7576  6                        !
;  7468     7577  7                        BEGIN
;  7469     7578  7                        IF DBG$DEF_SYM_FIND (LAST_OPERAND [TOKEN$B_LENGTH],
;  7470     7579  7                                             KIND, PRIMPTR,
;  7471     7580  7                                             DUMMY, DUMMY)
;  7472     7581  7                        THEN
;  7473     7582  8                            BEGIN
;  7474     7583  8                            IF .KIND EQL DEFINE_ADDRESS
;  7475     7584  8                            OR .KIND EQL DEFINE_VALUE
;  7476     7585  8                            THEN
;  7477     7586  9                                BEGIN
;  7478     7587  9
;  7479     7588  9
;  7480     7589  9                                ! We have found a matching DEFINEd symbol.
;  7481     7590  9                                ! Copy the descriptor into temporary memory.
;  7482     7591  9                                ! (Fourth parameter FALSE <-> copy into tempmem).
;  7483     7592  9                                !
;  7484     7593  9                                DBG$NCOPY_DESC (.PRIMPTR, PRIMPTR,
;  7485     7594  9                                                DUMMY, FALSE);
;  7486     7595  9                                EXITLOOP;
;  7487     7596  8                                END;
;  7488     7597  8
;  7489     7598  7                            END;
;  7490     7599  7
;  7491     7600  6                        END;
;  7492     7601  6
;  7493     7602  6                APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, NOT_REC_COMP);
```

```
;  7494          7603  6
;  7495          7604  6                    ! Call GETSYMBOL directly since all we need is the typeid.
;  7496          7605  6                    ! TRUE is passed in to tell GETSYMBOL that we do want it
;  7497          7606  6                    ! to look up symbol-types as well as the normal data-types.
;  7498          7607  6                    !
;  7499          7608  6                    DBG$STA_GETSYMBOL(PATHDESC, TYPEID, KIND, 0, 0, 0, TRUE);
;  7500          7609  6
;  7501          7610  6                    ! Check the output from getsymbol to see if a unique symbol
;  7502          7611  6                    ! was found.  If not, signal the appropriate error.
;  7503          7612  6                    !
;  7504          7613  6                    IF .TYPEID EQL 0
;  7505          7614  6                    THEN
;  7506          7615  7                        BEGIN
;  7507          7616  7                        DBG$NPATHDESC_TO_CS(PATHDESC, PATHSTRING);
;  7508          7617  7                        IF .KIND EQL RST$K_NOTUNIQUE
;  7509          7618  7                        THEN
;  7510          7619  7                            SIGNAL(DBG$_NOUNIQUE, 1, .PATHSTRING)
;  7511          7620  7                        ELSE
;  7512          7621  7                            IF .KIND EQL RST$K_OVERLOAD
;  7513          7622  7                            THEN
;  7514          7623  7                                SIGNAL(DBG$_NOTUNQOVR, 1, .PATHSTRING)
;  7515          7624  7                            ELSE
;  7516          7625  7                                SIGNAL(DBG$_NOSYMBOL, 1, .PATHSTRING);
;  7517          7626  6                        END;
;  7518          7627  6
;  7519          7628  6                    PRIMPTR = DBG$EVAL_ADA_TICK(.TYPEID, .TOKEN);
;  7520          7629  6
;  7521          7630  6                    ! Return the fact that we have built the result by now
;  7522          7631  6                    ! and that it is a value descriptor.  Because of the
;  7523          7632  6                    ! way the Ada tick tokens come through the above
;  7524          7633  6                    ! initialization code, this value was never set.
;  7525          7634  6                    ! And It Better Be!
;  7526          7635  6                    !
;  7527          7636  6                    RET_OPERAND_FLAG[0] = TRUE;
;  7528          7637  6
;  7529          7638  6                    END
;  7530          7639  5                ELSE
;  7531          7640  5                    SIGNAL(DBG$_SYNERREXPR, 1, .LAST_OPERAND[TOKEN$B_LENGTH]);
;  7532          7641  5
;  7533          7642  5                EXITLOOP;
;  7534          7643  4                END;
;  7535          7644  4
;  7536          7645  4
;  7537          7646  4            ! Handle the terminator operator immediately after the start of
;  7538          7647  4            ! the symbol.  The symbol thus consists of just a single name
;  7539          7648  4            ! or a constant.
;  7540          7649  4            !
;  7541          7650  4            [PRIMARY$K_ACT_START_TERM]:
;  7542          7651  5                BEGIN
;  7543          7652  5
;  7544          7653  5
;  7545          7654  5                ! If the last operand was an identifier, we append it to
;  7546          7655  5                ! the current Pathname Descriptor and convert that to a
;  7547          7656  5                ! Primary Descriptor.
;  7548          7657  5                !
;  7549          7658  5                IF .LAST_OPERAND[TOKEN$W_CODE] EQL TOKEN$K_IDENTIFIER
;  7550          7659  5                THEN
```

```
;  7551     7660  6                BEGIN
;  7552     7661  6
;  7553     7662  6
;  7554     7663  6                ! First check for DEFINEd symbols.
;  7555     7664  6                ! Check that no invocation number is present.
;  7556     7665  6                !
;  7557     7666  6                IF .PATHDESC[PTH$B_LOCINVOC] EQL 0
;  7558     7667  6                THEN
;  7559     7668  6
;  7560     7669  6                    ! Look up the symbol in the DEFINE symbol table.
;  7561     7670  6                    !
;  7562     7671  6                    BEGIN
;  7563     7672  7                    IF DBG$DEF_SYM_FIND (LAST_OPERAND [TOKEN$B_LENGTH],
;  7564     7673  7                                         KIND, PRIMPTR,
;  7565     7674  7                                         DUMMY, DUMMY)
;  7566     7675  7                    THEN
;  7567     7676  7                        BEGIN
;  7568     7677  8                        IF .KIND EQL DEFINE_ADDRESS
;  7569     7678  8                        OR .KIND EQL DEFINE_VALUE
;  7570     7679  8                        THEN
;  7571     7680  8                            BEGIN
;  7572     7681  9
;  7573     7682  9                            ! We have found a matching DEFINEd symbol.
;  7574     7683  9                            ! Copy the descriptor into temporary memory.
;  7575     7684  9                            ! (Fourth parameter FALSE <-> copy into tempmem).
;  7576     7685  9                            !
;  7577     7686  9                            DBG$NCOPY_DESC (.PRIMPTR, PRIMPTR,
;  7578     7687  9                                            DUMMY, FALSE);
;  7579     7688  9                            EXITLOOP;
;  7580     7689  9                            END;
;  7581     7690  9
;  7582     7691  8                        END;
;  7583     7692  8
;  7584     7693  7                    END;
;  7585     7694  7
;  7586     7695  6                ;
;  7587     7696  6
;  7588     7697  6                APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, NOT_REC_COMP);
;  7589     7698  6                PRIMPTR = PATHNAME_TO_PRIMARY(PATHDESC,SUBSCR_DESC,
;  7590     7699  6                                              .PLIPTR,SAVED_PATHDESC);
;  7591     7700  6                END
;  7592     7701  6
;  7593     7702  6
;  7594     7703  6                ! Otherwise, the last operand was a constant of some
;  7595     7704  6                ! sort so we create a Value Descriptor for it.
;  7596     7705  6                !
;  7597     7706  5                ELSE
;  7598     7707  5                    PRIMPTR = CONSTANT_TO_VALDESCR(.LAST_OPERAND);
;  7599     7708  5
;  7600     7709  5
;  7601     7710  5                ! Exit from the parse loop (the get-token loop).
;  7602     7711  5                !
;  7603     7712  5                EXITLOOP;
;  7604     7713  4                END;
;  7605     7714  4
;  7606     7715  4
;  7607     7716  4            ! Handle the backslash operator after a previous backslash ope-
```

```
;  7608     7717  4                    ! rator (e.g., A\B\C).  Note that this is not allowed after a
;  7609     7718  4                    ! numeric pathname.  Also handle a backslash operator after an
;  7610     7719  4                    ! invocation number.  Append the last operand (which must be an
;  7611     7720  4                    ! identifier) to the current Pathname Descriptor.
;  7612     7721  4                    !
;  7613     7722  4                    [PRIMARY$K_ACT_SLASH_SLASH]:
;  7614     7723  5                        BEGIN
;  7615     7724  5                        IF .NUMERIC_PATHNAME
;  7616     7725  5                        THEN
;  7617     7726  5                            SIGNAL(DBG$_ILLPATHELEM, 1, LAST_OPERAND[TOKEN$B_LENGTH]);
;  7618     7727  5
;  7619     7728  5                        APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, NOT_REC_COMP);
;  7620     7729  4                        END;
;  7621     7730  4
;  7622     7731  4
;  7623     7732  4                    ! Handle a dot (data qualification) after a backslash.  Here we
;  7624     7733  4                    ! complete the current Pathname Descriptor and convert it to a
;  7625     7734  4                    ! Primary Descriptor.
;  7626     7735  4                    !
;  7627     7736  4                    [PRIMARY$K_ACT_SLASH_DOT]:
;  7628     7737  5                        BEGIN
;  7629     7738  5                        APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, NOT_REC_COMP);
;  7630     7739  5                        PRIMPTR = PATHNAME_TO_PRIMARY(PATHDESC,SUBSCR_DESC,
;  7631     7740  5                                                      .PLIPTR,SAVED_PATHDESC);
;  7632     7741  4                        END;
;  7633     7742  4
;  7634     7743  4
;  7635     7744  4                    ! Handle a dot after a backslash. In PLI we do not call
;  7636     7745  4                    ! PATHNAME_TO_PRIMARY until after collecting all the record
;  7637     7746  4                    ! components.
;  7638     7747  4                    !
;  7639     7748  4                    [PRIMARY$K_ACT_SLASH_DOT_PLI]:
;  7640     7749  5                        BEGIN
;  7641     7750  5                        APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, NOT_REC_COMP);
;  7642     7751  4                        END;
;  7643     7752  4
;  7644     7753  4
;  7645     7754  4                    ! Handle a subscript parenthesis after a backslash.  Here we
;  7646     7755  4                    ! complete the current Pathname Descriptor and convert it to a
;  7647     7756  4                    ! Primary Descriptor.  We then pick up all the subscript expres-
;  7648     7757  4                    ! sions.
;  7649     7758  4                    !
;  7650     7759  4                    [PRIMARY$K_ACT_SLASH_SUBSCR]:
;  7651     7760  5                        BEGIN
;  7652     7761  5                        APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, NOT_REC_COMP);
;  7653     7762  5                        PRIMPTR = PATHNAME_TO_PRIMARY(PATHDESC,SUBSCR_DESC,
;  7654     7763  5                                                      .PLIPTR, SAVED_PATHDESC, TRUE);
;  7655     7764  5                        GET_SUBSCRIPTS(.PRIMPTR);
;  7656     7765  4                        END;
;  7657     7766  4
;  7658     7767  4
;  7659     7768  4                    ! Handle a subscript after a backslash in language PLI. In PLI
;  7660     7769  4                    ! we do not incorporate the subscripts into the Primary
;  7661     7770  4                    ! Descriptor until later.
;  7662     7771  4                    !
;  7663     7772  4                    [PRIMARY$K_ACT_SLASH_SUBSCR_PLI]:
;  7664     7773  5                        BEGIN
```

```
;  7665          7774  5                  APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, NOT_REC_COMP);
;  7666          7775  5                  SAVE_SUBSCRIPTS(PATHDESC, SUBSCR_DESC);
;  7667          7776  4                  END;
;  7668          7777  4
;  7669          7778  4
;  7670          7779  4                ! Handle a subscript left bracket after a backslash for language
;  7671          7780  4                ! BLISS.  Here we complete the current Pathname Descriptor and
;  7672          7781  4                ! convert it to a Primary Descriptor.  We then pick up all the
;  7673          7782  4                ! subscript expressions.  BLISS has to be treated separately
;  7674          7783  4                ! because it represents data differently in the RST and DST.
;  7675          7784  4                !
;  7676          7785  4                [PRIMARY$K_ACT_SLASH_SUBSCR_BLI]:
;  7677          7786  5                    BEGIN
;  7678          7787  5                    APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, NOT_REC_COMP);
;  7679          7788  5                    PRIMPTR = PATHNAME_TO_PRIMARY(PATHDESC,SUBSCR_DESC,
;  7680          7789  5                                                  .PLIPTR,SAVED_PATHDESC);
;  7681          7790  5                    GET_BLISS_SUBSCRIPTS(.PRIMPTR, LAST_OPERAND[TOKEN$B_LENGTH]);
;  7682          7791  4                    END;
;  7683          7792  4
;  7684          7793  4
;  7685          7794  4                ! Handle the PASCAL dereference operator (^) occuring after
;  7686          7795  4                ! a pathname qualified name.  Append the last identifier onto
;  7687          7796  4                ! the pathname.  Then build the pathname into a Primary
;  7688          7797  4                ! Descriptor.  Then call the routine that handles derefencing
;  7689          7798  4                ! --it just lights the EVAL bit on the sub-node and then
;  7690          7799  4                ! allocates a new subnode for the object being pointed to.
;  7691          7800  4                !
;  7692          7801  4                [PRIMARY$K_ACT_SLASH_DEREF]:
;  7693          7802  5                    BEGIN
;  7694          7803  5                    APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, NOT_REC_COMP);
;  7695          7804  5                    PRIMPTR = PATHNAME_TO_PRIMARY(PATHDESC,SUBSCR_DESC,
;  7696          7805  5                                                  .PLIPTR,SAVED_PATHDESC);
;  7697          7806  5                    GET_DEREFERENCE(.PRIMPTR);
;  7698          7807  4                    END;
;  7699          7808  4
;  7700          7809  4
;  7701          7810  4                ! Handle a PLI dereference as in A\B->C. Here we convert the
;  7702          7811  4                ! left-hand-side to a Primary, and then save away the Primary.
;  7703          7812  4                ! The parse is then "backed up" to the start state to pick
;  7704          7813  4                ! up the right hand side of the "->". Later, in the
;  7705          7814  4                ! PATHNAME_TO_PRIMARY routine, the two primaries are glued
;  7706          7815  4                ! together.
;  7707          7816  4                !
;  7708          7817  4                [PRIMARY$K_ACT_SLASH_DEREF_PLI]:
;  7709          7818  5                    BEGIN
;  7710          7819  5
;  7711          7820  5                    ! Obtain value of expression to left of "->".
;  7712          7821  5                    !
;  7713          7822  5                    APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, NOT_REC_COMP);
;  7714          7823  5                    PLIPTR = PATHNAME_TO_PRIMARY(PATHDESC,SUBSCR_DESC,
;  7715          7824  5                                                 .PLIPTR,SAVED_PATHDESC);
;  7716          7825  5                    IF (.PLIPTR[DBG$B_DHDR_KIND] NEQ RST$K_DATA) OR
;  7717          7826  6                       (.PLIPTR[DBG$B_DHDR_FCODE] NEQ RST$K_TYPE_PTR)
;  7718          7827  5                    THEN
;  7719          7828  6                        BEGIN
;  7720          7829  6                        LOCAL
;  7721          7830  6                            NAME;
```

```
;  7722          7831  6                 DBG$NPATHDESC_TO_CS(PATHDESC, NAME);
;  7723          7832  6                 SIGNAL(DBG$_VALNOTADDR, 1, .NAME);
;  7724          7833  5                 END;
;  7725          7834  5
;  7726          7835  5             ! Re-initialize.
;  7727          7836  5             !
;  7728          7837  5             PRIMPTR = 0;
;  7729          7838  5             NUMERIC_PATHNAME = FALSE;
;  7730          7839  5             CH$FILL(0, DBG$K_PATHNAME_SIZE*%UPVAL, PATHDESC);
;  7731          7840  5             PATHVECTOR = PATHDESC[PTH$A_PATHVECTOR];
;  7732          7841  5             CH$FILL(0, SUBSCR_DESC_SIZE, SUBSCR_DESC);
;  7733          7842  4             END;
;  7734          7843  4
;  7735          7844  4
;  7736          7845  4         ! Handle the Ada tick operator after a backslash.  Here we
;  7737          7846  4         ! just complete the Pathname Descriptor and convert it to a
;  7738          7847  4         ! Primary Descriptor.  We are then done with the symbol.
;  7739          7848  4         !
;  7740          7849  4         [PRIMARY$K_ACT_SLASH_TICK]:
;  7741          7850  5             BEGIN
;  7742          7851  5             APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, NOT_REC_COMP);
;  7743          7852  5
;  7744          7853  5             ! Call GETSYMBOL directly since all we need is the typeid.
;  7745          7854  5             ! TRUE is passed in to tell GETSYMBOL that we do want it
;  7746          7855  5             ! to look up symbol-types as well as the normal data-types.
;  7747          7856  5             !
;  7748          7857  5             DBG$STA_GETSYMBOL(PATHDESC, TYPEID, KIND, 0, 0, 0, TRUE);
;  7749          7858  5
;  7750          7859  5             ! Check the output from getsymbol to see if a unique symbol
;  7751          7860  5             ! was found.  If not, signal the appropriate error.
;  7752          7861  5             !
;  7753          7862  5             IF .TYPEID EQL 0
;  7754          7863  5             THEN
;  7755          7864  6                 BEGIN
;  7756          7865  6                 DBG$NPATHDESC_TO_CS(PATHDESC, PATHSTRING);
;  7757          7866  6                 IF  KIND EQL RST$K_NOTUNIQUE
;  7758          7867  6                 THEN
;  7759          7868  6                     SIGNAL(DBG$_NOUNIQUE, 1, .PATHSTRING)
;  7760          7869  6                 ELSE
;  7761          7870  6                     IF .KIND EQL RST$K_OVERLOAD
;  7762          7871  6                     THEN
;  7763          7872  6                         SIGNAL(DBG$_NOTUNQOVR, 1, .PATHSTRING)
;  7764          7873  6                     ELSE
;  7765          7874  6                         SIGNAL(DBG$_NOSYMBOL, 1, .PATHSTRING);
;  7766          7875  5                 END;
;  7767          7876  5
;  7768          7877  5             PRIMPTR = DBG$EVAL_ADA_TICK(.TYPEID, .TOKEN);
;  7769          7878  5
;  7770          7879  5             ! Return the fact that we have built the result by now
;  7771          7880  5             ! and that it is a value descriptor.  Because of the
;  7772          7881  5             ! way the Ada tick tokens come through the above
;  7773          7882  5             ! initialization code, this value was never set.
;  7774          7883  5             ! And It Better Be!
;  7775          7884  5             !
;  7776          7885  5             RET_OPERAND_FLAG[0] = TRUE;
;  7777          7886  5
;  7778          7887  5             EXITLOOP;
```

```
;  7779          7888  4                    END;
;  7780          7889  4
;  7781          7890  4
;  7782          7891  4                ! Handle the terminator operator after a backslash.  Here we
;  7783          7892  4                ! just complete the Pathname Descriptor and convert it to a
;  7784          7893  4                ! Primary Descriptor.  We are then done with the symbol.
;  7785          7894  4                !
;  7786          7895  4                [PRIMARY$K_ACT_SLASH_TERM]:
;  7787          7896  5                    BEGIN
;  7788          7897  5                    APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, NOT_REC_COMP);
;  7789          7898  5                    PRIMPTR = PATHNAME_TO_PRIMARY(PATHDESC,SUBSCR_DESC,
;  7790          7899  5                                                  .PLIPTR,SAVED_PATHDESC);
;  7791          7900  5                    EXITLOOP;
;  7792          7901  4                    END;
;  7793          7902  4
;  7794          7903  4
;  7795          7904  4                ! Handle a slash after an "OF" in COBOL.
;  7796          7905  4                !
;  7797          7906  4                [PRIMARY$K_ACT_DOT_SLASH_COB]:
;  7798          7907  5                    BEGIN
;  7799          7908  5                    APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, NOT_REC_COMP);
;  7800          7909  4                    END;
;  7801          7910  4
;  7802          7911  4                ! Handle a dot (data qualification) after another dot.  Check
;  7803          7912  4                ! that the last operand is an identifier and that it is a valid
;  7804          7913  4                ! component name of the current record type.  Get its SYMID,
;  7805          7914  4                ! etc., and add it to the Primary Descriptor being built.
;  7806          7915  4                !
;  7807          7916  4                [PRIMARY$K_ACT_DOT_DOT]:
;  7808          7917  4                    GET_RECORD_COMPONENT(.PRIMPTR, LAST_OPERAND[TOKEN$B_LENGTH]);
;  7809          7918  4
;  7810          7919  4
;  7811          7920  4                ! Handle a dot after another dot in PLI. In PLI we do not
;  7812          7921  4                ! call PATHNAME_TO_PRIMARY until after collecting all
;  7813          7922  4                ! the record components.
;  7814          7923  4                !
;  7815          7924  4                [PRIMARY$K_ACT_DOT_DOT_PLI]:
;  7816          7925  5                    BEGIN
;  7817          7926  5                    APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, REC_COMP);
;  7818          7927  4                    END;
;  7819          7928  4
;  7820          7929  4
;  7821          7930  4                ! Handle an "OF" operator after another "OF" operator
;  7822          7931  4                ! COBOL.
;  7823          7932  4                !
;  7824          7933  4                [PRIMARY$K_ACT_DOT_DOT_COB]:
;  7825          7934  5                    BEGIN
;  7826          7935  5                    APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, COB_REC_COMP);
;  7827          7936  4                    END;
;  7828          7937  4
;  7829          7938  4                ! Handle subscripting after a dot (data qualification).  Check
;  7830          7939  4                ! that the last operand is an identifier and that it is a valid
;  7831          7940  4                ! component name of the current record type.  Get its SYMID,
;  7832          7941  4                ! etc., and add it to the Primary Descriptor being built.  Then
;  7833          7942  4                ! pick up all the subscript expressions and add their values to
;  7834          7943  4                ! the Primary Descriptor.
;  7835          7944  4                !
```

```
; 7836        7945  4          [PRIMARY$K_ACT_DOT_SUBSCR]:
; 7837        7946  5              BEGIN
; 7838        7947  5              GET_RECORD_COMPONENT(.PRIMPTR, LAST_OPERAND[TOKEN$B_LENGTH]);
; 7839        7948  5              GET_SUBSCRIPTS(.PRIMPTR);
; 7840        7949  4              END;
; 7841        7950  4
; 7842        7951  4
; 7843        7952  4          ! Handle a subscript after a dot in PLI. In PLI we do not
; 7844        7953  4          ! call PATHNAME_TO_PRIMARY until after picking up all
; 7845        7954  4          ! the record components.
; 7846        7955  4          !
; 7847        7956  4          [PRIMARY$K_ACT_DOT_SUBSCR_PLI]:
; 7848        7957  5              BEGIN
; 7849        7958  5              APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, REC_COMP);
; 7850        7959  5              SAVE_SUBSCRIPTS(PATHDESC, SUBSCR_DESC);
; 7851        7960  4              END;
; 7852        7961  4
; 7853        7962  4
; 7854        7963  4          ! Handle the subscript operator after the "OF" operator
; 7855        7964  4          ! in COBOL.
; 7856        7965  4          !
; 7857        7966  4          [PRIMARY$K_ACT_DOT_SUBSCR_COB]:
; 7858        7967  5              BEGIN
; 7859        7968  5              APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, NOT_REC_COMP);
; 7860        7969  5              SAVE_SUBSCRIPTS(PATHDESC, SUBSCR_DESC);
; 7861        7970  4              END;
; 7862        7971  4
; 7863        7972  4          ! Handle the PASCAL dereference operator (^) occurring after
; 7864        7973  4          ! a dot (data qualification).  Here we call a routine which
; 7865        7974  4          ! appends the last operand (representing a record component)
; 7866        7975  4          ! onto the Primary Descriptor being built.  We then call the
; 7867        7976  4          ! routine that handles derefencing--it just lights the EVAL
; 7868        7977  4          ! bit on the sub-node and then  allocates a new subnode for
; 7869        7978  4          ! the object being pointed to.
; 7870        7979  4          !
; 7871        7980  4          [PRIMARY$K_ACT_DOT_DEREF]:
; 7872        7981  5              BEGIN
; 7873        7982  5              GET_RECORD_COMPONENT(.PRIMPTR, LAST_OPERAND[TOKEN$B_LENGTH]);
; 7874        7983  5              GET_DEREFERENCE(.PRIMPTR);
; 7875        7984  4              END;
; 7876        7985  4
; 7877        7986  4
; 7878        7987  4          ! Handle a PLI dereference as in A.B->C. Here we convert the
; 7879        7988  4          ! left-hand-side to a Primary, and then save away the Primary.
; 7880        7989  4          ! We loop back to the start state to pick up the rest
; 7881        7990  4          ! of the expression.
; 7882        7991  4          !
; 7883        7992  4          [PRIMARY$K_ACT_DOT_DEREF_PLI]:
; 7884        7993  5              BEGIN
; 7885        7994  5
; 7886        7995  5              ! Save away the Primary constructed so far.
; 7887        7996  5              !
; 7888        7997  5              APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, REC_COMP);
; 7889        7998  5              PLIPTR = PATHNAME_TO_PRIMARY(PATHDESC,SUBSCR_DESC,
; 7890        7999  5                                           .PLIPTR,SAVED_PATHDESC);
; 7891        8000  5              IF (.PLIPTR[DBG$B_DHDR_KIND] NEQ RST$K_DATA) OR
; 7892        8001  6                 (.PLIPTR[DBG$B_DHDR_FCODE] NEQ RST$K_TYPE_PTR)
```

```
;  7893          8002  5                 THEN
;  7894          8003  6                     BEGIN
;  7895          8004  6                     LOCAL
;  7896          8005  6                         NAME;
;  7897          8006  6                     DBG$NPATHDESC_TO_CS(PATHDESC, NAME);
;  7898          8007  6                     SIGNAL(DBG$_VALNOTADDR, 1, .NAME);
;  7899          8008  5                     END;
;  7900          8009  5
;  7901          8010  5                     ! Re-initialize.
;  7902          8011  5                     !
;  7903          8012  5                     PRIMPTR = 0;
;  7904          8013  5                     NUMERIC_PATHNAME = FALSE;
;  7905          8014  5                     CH$FILL(0, DBG$K_PATHNAME_SIZE+%UPVAL, PATHDESC);
;  7906          8015  5                     PATHVECTOR = PATHDESC[PTH$A_PATHVECTOR];
;  7907          8016  5                     CH$FILL(0, SUBSCR_DESC_SIZE, SUBSCR_DESC);
;  7908          8017  4                     END;
;  7909          8018  4
;  7910          8019  4
;  7911          8020  4                 ! Handle the Ada tick operator after a dot (data qualifica-
;  7912          8021  4                 ! tion).  Call the routine that looks up the SYMID for the
;  7913          8022  4                 ! record component and adds that to the Primary Descriptor.
;  7914          8023  4                 ! Then exit the parse loop.
;  7915          8024  4                 !
;  7916          8025  4                 [PRIMARY$K_ACT_DOT_TICK]:
;  7917          8026  5                     BEGIN
;  7918          8027  5                     APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, REC_COMP);
;  7919          8028  5
;  7920          8029  5                     ! Call GETSYMBOL directly since all we need is the typeid.
;  7921          8030  5                     ! TRUE is passed in to tell GETSYMBOL that we do want it
;  7922          8031  5                     ! to look up symbol-types as well as the normal data-types.
;  7923          8032  5                     !
;  7924          8033  5                     DBG$STA_GETSYMBOL(PATHDESC, TYPEID, KIND, 0, 0, 0, TRUE);
;  7925          8034  5
;  7926          8035  5                     ! Check the output from getsymbol to see if a unique symbol
;  7927          8036  5                     ! was found.  If not, signal the appropriate error.
;  7928          8037  5                     !
;  7929          8038  5                     IF .TYPEID EQL 0
;  7930          8039  5                     THEN
;  7931          8040  6                         BEGIN
;  7932          8041  6                         DBG$NPATHDESC_TO_CS(PATHDESC, PATHSTRING);
;  7933          8042  6                         IF .KIND EQL RST$K_NOTUNIQUE
;  7934          3043  6                         THEN
;  7935          8044  6                             SIGNAL(DBG$_NOUNIQUE, 1, .PATHSTRING)
;  7936          8045  6                         ELSE
;  7937          8046  6                             IF .KIND EQL RST$K_OVERLOAD
;  7938          8047  6                             THEN
;  7939          8048  6                                 SIGNAL(DBG$_NOTUNQOVR, 1, .PATHSTRING)
;  7940          8049  6                             ELSE
;  7941          8050  6                                 SIGNAL(DBG$_NOSYMBOL, 1, .PATHSTRING);
;  7942          8051  5                         END;
;  7943          8052  5
;  7944          8053  5                     PRIMPTR = DBG$EVAL_ADA_TICK(.TYPEID, .TOKEN);
;  7945          8054  5
;  7946          8055  5                     ! Return the fact that we have built the result by now
;  7947          8056  5                     ! and that it is a value descriptor.  Because of the
;  7948          8057  5                     ! way the Ada tick tokens come through the above
;  7949          8058  5                     ! initialization code, this value was never set.
```

```
;  7950     8059  5                ! And It Better Be!
;  7951     8060  5                !
;  7952     8061  5                RET_OPERAND_FLAG[0] = TRUE;
;  7953     8062  5
;  7954     8063  5                EXITLOOP;
;  7955     8064  4                END;
;  7956     8065  4
;  7957     8066  4
;  7958     8067  4             ! Handle the terminator operator after a dot (data qualifica-
;  7959     8068  4             ! tion).  Call the routine that looks up the SYMID for the
;  7960     8069  4             ! record component and adds that to the Primary Descriptor.
;  7961     8070  4             ! Then exit the parse loop.
;  7962     8071  4             !
;  7963     8072  4             [PRIMARY$K_ACT_DOT_TERM]:
;  7964     8073  5                BEGIN
;  7965     8074  5                GET_RECORD_COMPONENT(.PRIMPTR, LAST_OPERAND[TOKEN$B_LENGTH]);
;  7966     8075  5                EXITLOOP;
;  7967     8076  4                END;
;  7968     8077  4
;  7969     8078  4
;  7970     8079  4             ! Handle the end of the Primary after a dot in PLI. At this
;  7971     8080  4             ! point we call PATHNAME_TO_PRIMARY, and then exit the loop.
;  7972     8081  4             !
;  7973     8082  4             [PRIMARY$K_ACT_DOT_TERM_PLI]:
;  7974     8083  5                BEGIN
;  7975     8084  5                APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, REC_COMP);
;  7976     8085  5                PRIMPTR = PATHNAME_TO_PRIMARY(PATHDESC,SUBSCR_DESC,
;  7977     8086  5                                              .PLIPTR, SAVED_PATHDESC);
;  7978     8087  5                EXITLOOP;
;  7979     8088  4                END;
;  7980     8089  4
;  7981     8090  4
;  7982     8091  4             ! Handle the end of the Primary after an "OF" operator
;  7983     8092  4             ! (record component selection) in COBOL.
;  7984     8093  4             !
;  7985     8094  4             [PRIMARY$K_ACT_DOT_TERM_COB]:
;  7986     8095  5                BEGIN
;  7987     8096  5                APPEND_TO_PATHNAME(PATHDESC, .LAST_OPERAND, NOT_REC_COMP);
;  7988     8097  5                PRIMPTR = PATHNAME_TO_PRIMARY(PATHDESC,SUBSCR_DESC,
;  7989     8098  5                                              .PLIPTR,SAVED_PATHDESC);
;  7990     8099  5                EXITLOOP;
;  7991     8100  4                END;
;  7992     8101  4
;  7993     8102  4             ! Handle a dot (data qualification) after subscripting.  We do
;  7994     8103  4             ! nothing with the dot until after we pick up the component
;  7995     8104  4             ! name. However, we do call a routine that fixes up the
;  7996     8105  4             ! bounds in the array subnode after subscripting has been
;  7997     8106  4             ! completed.
;  7998     8107  4             !
;  7999     8108  4             [PRIMARY$K_ACT_SUBSCR_DOT]:
;  8000     8109  4                FIX_UP_PRIMARY(.PRIMPTR);
;  8001     8110  4
;  8002     8111  4
;  8003     8112  4             ! Handle a dot after a subscript in PLI. There is actually
;  8004     8113  4             ! nothing to do here.
;  8005     8114  4             !
;  8006     8115  4             [PRIMARY$K_ACT_SUBSCR_DOT_PLI]:
```

```
;  8007     8116  5                    BEGIN
;  8008     8117  5                    0;
;  8009     8118  4                    END;
;  8010     8119  4
;  8011     8120  4
;  8012     8121  4                ! Handle subscripting parentheses immediately after a previous
;  8013     8122  4                ! instance of subscripting.  Just do the subscripting normally.
;  8014     8123  4                ! Pick up the subscript expressions and add their values to the
;  8015     8124  4                ! Primary Descriptor being built.
;  8016     8125  4                !
;  8017     8126  4                [PRIMARY$K_ACT_SUBSCR_SUBSCR]:
;  8018     8127  4                    GET_SUBSCRIPTS(.PRIMPTR);
;  8019     8128  4
;  8020     8129  4
;  8021     8130  4                ! Handle a subscript immediately after another subscript
;  8022     8131  4                ! in PLI. In PLI we save up the subscripts until after we
;  8023     8132  4                ! collect a full pathname including record components.
;  8024     8133  4                !
;  8025     8134  4                [PRIMARY$K_ACT_SUBSCR_SUBSCR_PLI]:
;  8026     8135  5                    BEGIN
;  8027     8136  5                    SAVE_SUBSCRIPTS(PATHDESC, SUBSCR_DESC);
;  8028     8137  4                    END;
;  8029     8138  4
;  8030     8139  4
;  8031     8140  4                ! Handle a PASCAL dereference operator (^) occuring after
;  8032     8141  4                ! a subscripting operation.  First call a routine that fixes
;  8033     8142  4                ! up the bounds in the array subnode after subscripting has
;  8034     8143  4                ! been  completed.  Then  call  the  routine  that  handles
;  8035     8144  4                ! derefencing--it just lights the EVAL bit on the sub-node and
;  8036     8145  4                ! then allocates a new subnode for the object being pointed to.
;  8037     8146  4                !
;  8038     8147  4                [PRIMARY$K_ACT_SUBSCR_DEREF]:
;  8039     8148  5                    BEGIN
;  8040     8149  5                    FIX_UP_PRIMARY(.PRIMPTR);
;  8041     8150  5                    GET_DEREFERENCE(.PRIMPTR);
;  8042     8151  4                    END;
;  8043     8152  4
;  8044     8153  4
;  8045     8154  4                ! Handle a PLI dereference as in A(2)->B. Here we convert the
;  8046     8155  4                ! left-hand-side to a Primary, and save away that Primary.
;  8047     8156  4                ! We loop back to the start state to pick up the rest
;  8048     8157  4                ! of the expression.
;  8049     8158  4                !
;  8050     8159  4                [PRIMARY$K_ACT_SUBSCR_DEREF_PLI]:
;  8051     8160  5                    BEGIN
;  8052     8161  5
;  8053     8162  5                    ! Save away the Primary obtained so far.
;  8054     8163  5                    !
;  8055     8164  5                    PLIPTR = PATHNAME_TO_PRIMARY(PATHDESC,SUBSCR_DESC,
;  8056     8165  5                                                 .PLIPTR, SAVED_PATHDESC);
;  8057     8166  5                    IF (.PLIPTR[DBG$B_DHDR_KIND] NEQ RST$K_DATA) OR
;  8058     8167  6                       (.PLIPTR[DBG$B_DHDR_FCODE] NEQ RST$K_TYPE_PTR)
;  8059     8168  5                    THEN
;  8060     8169  6                        BEGIN
;  8061     8170  6                        LOCAL
;  8062     8171  6                            NAME;
;  8063     8172  6                        DBG$NPATHDESC_TO_CS(PATHDESC, NAME);
```

```
;  8064          8173  6                 SIGNAL(DBG$_VALNOTADDR, 1, .NAME);
;  8065          8174  5                 END;
;  8066          8175  5
;  8067          8176  5             ! Re-initialize.
;  8068          8177  5             !
;  8069          8178  5             PRIMPTR = 0;
;  8070          8179  5             NUMERIC_PATHNAME = FALSE;
;  8071          8180  5             CH$FILL(0, DBG$K_PATHNAME_SIZE*%UPVAL, PATHDESC);
;  8072          8181  5             PATHVECTOR = PATHDESC[PTH$A_PATHVECTOR];
;  8073          8182  5             CH$FILL(0, SUBSCR_DESC_SIZE, SUBSCR_DESC);
;  8074          8183  4             END;
;  8075          8184  4
;  8076          8185  4
;  8077          8186  4         ! Handle subscripting followed by the terminator operator.
;  8078          8187  4         ! First call a routine that fixes up the bounds in the array
;  8079          8188  4         ! subnode after subscripting has been completed.  The
;  8080          8189  4         ! Primary Descriptor is now complete, so we exit the parse
;  8081          8190  4         ! loop.
;  8082          8191  4         !
;  8083          8192  4         [PRIMARY$K_ACT_SUBSCR_TERM]:
;  8084          8193  5             BEGIN
;  8085          8194  5             FIX_UP_PRIMARY(.PRIMPTR);
;  8086          8195  5             EXITLOOP;
;  8087          8196  4             END;
;  8088          8197  4
;  8089          8198  4
;  8090          8199  4         ! Handle a subscript followed by the end of the expression
;  8091          8200  4         ! in PLI.
;  8092          8201  4         !
;  8093          8202  4         [PRIMARY$K_ACT_SUBSCR_TERM_PLI]:
;  8094          8203  5             BEGIN
;  8095          8204  5             PRIMPTR = PATHNAME_TO_PRIMARY(PATHDESC,SUBSCR_DESC,
;  8096          8205  5                                           .PLIPTR, SAVED_PATHDESC);
;  8097          8206  5             EXITLOOP;
;  8098          8207  4             END;
;  8099          8208  4
;  8100          8209  4
;  8101          8210  4
;  8102          8211  4         ! Handle a PASCAL dereference operator (^) followed by a dot.
;  8103          8212  4         ! (E.g. "A^.B").  We actually do nothing here since the deref-
;  8104          8213  4         ! erence operator was handled already, and the dot operator is
;  8105          8214  4         ! not handled until after we pick up the component.
;  8106          8215  4         !
;  8107          8216  4         [PRIMARY$K_ACT_DEREF_DOT]:
;  8108          8217  4             0;
;  8109          8218  4
;  8110          8219  4
;  8111          8220  4         ! Handle a PASCAL dereference operator (^) followed by a
;  8112          8221  4         ! subscript.  Here we just pick up the subscripts.
;  8113          8222  4         !
;  8114          8223  4         [PRIMARY$K_ACT_DEREF_SUBSCR]:
;  8115          8224  4             GET_SUBSCRIPTS(.PRIMPTR);
;  8116          8225  4
;  8117          8226  4
;  8118          8227  4         ! Handle a PASCAL dereference operator (^) followed by another
;  8119          8228  4         ! dereference.  We call the routine that handles derefencing--
;  8120          8229  4         ! it just lights the EVAL bit on the sub-node and then allo-
```

```
;  8121          8230  4                 ! cates a new subnode for the object being pointed to.
;  8122          8231  4                 !
;  8123          8232  4                 [PRIMARY$K_ACT_DEREF_DEREF]:
;  8124          8233  4                     GET_DEREFERENCE(.PRIMPTR);
;  8125          8234  4
;  8126          8235  4
;  8127          8236  4                 ! Handle a PASCAL dereference operator (^) followed by a termi-
;  8128          8237  4                 ! nator.  The Primary Descriptor is now complete, so we exit
;  8129          8238  4                 ! the parse loop.
;  8130          8239  4                 !
;  8131          8240  4                 [PRIMARY$K_ACT_DEREF_TERM]:
;  8132          8241  4                     EXITLOOP;
;  8133          8242  4
;  8134          8243  4
;  8135          8244  4                 ! Any other CASE index constitutes an internal DEBUG error.
;  8136          8245  4                 !
;  8137          8246  4                 [INRANGE, OUTRANGE]:
;  8138          8247  4                     $DBG_ERROR('DBGPARSER\PRIMARY_PARSER 10');
;  8139          8248  4
;  8140          8249  4                 TES;
;  8141          8250  4
;  8142          8251  3             END;                            ! End of ELSE-clause for operators
;  8143          8252  3
;  8144          8253  2         END;                                ! End of the get-token loop
;  8145          8254  2
;  8146          8255  2
;  8147          8256  2     ! We are all done parsing the primary. Return a pointer to the
;  8148          8257  2     ! descriptor we have constructed.
;  8149          8258  2     !
;  8150          8259  2     RET_TOKEN[0] = .PRIMPTR;
;  8151          8260  2     RETURN;
;  8152          8261  2
;  8153          8262  1     END;


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

                                                    4C  30  031A0 TOKEN_IS_INTEGER:
                                                                .BYTE   48, 76                                  ;
                                                            031A2       .BLKB   1
                                                        00  031A3 P.AXG:    .BYTE   0                           ;
                                                        00  031A4 P.AXH:    .BYTE   0                           ;
                                            20  6C  61  6D  69  63  65  64  08  031A5 P.AXI:    .ASCII  <8>\decimal \               ;
67  65  74  6E  69  20  64  72  6F  77  67  6E  6F  6C  10  031AE P.AXJ:    .ASCII  <16>\longword integer\      ;
                                                        72  65  031BD
4D  49  52  50  5C  52  45  53  52  41  50  47  42  44  1B  031BF P.AXK:    .ASCII  <27>\DBGPARSER\<92>\PRIMARY_PARSER 10\      ;
        30  31  20  52  45  53  52  41  50  5F  59  52  41  031CE                                               ;


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

                                                        0FFC 00000          .ENTRY  DBG$PRIMARY_PARSER, Save R2,R3,R4,R5,R6,R7,-; 6691
                                                                            R8,R9,R10,R11                       ;
                                                5E      FBA8    CE  9E 00002        MOVAB   -1112(SP), SP                       ;
                                                50 00000000'    EF  D0 00007        MOVL    SAVED_TOKEN, R0                     ; 6851
```

```
                                 06              6C  91 0000E            CMPB    (AP), #6                                        ; 6840
                                                 1D  1B 00011            BLEQU   2$                                              ;
                                                 50  D5 00013            TSTL    R0                                              ; 6851
                                                 06  13 00015            BEQL    1$                                              ;
                                14   BC      1C   AC  D0 00017            MOVL    28(AP), @RET_TOKEN                              ; 6854
                                                 04  00 0001C            RET                                                     ; 6853
                                56               01  CE 0001D 1$:        MNEGL   #1, LAST_OPERAND                                ; 6867
                                10   AE      04   AC  D0 00020            MOVL    OPERAND_EXPECTED_FLAG, OPERAND_EXPECTED         ; 6868
                                59      20       AC  D0 00025            MOVL    32(AP), STATE_INDEX                             ; 6869
                                24   AE      1C   AC  D0 00029            MOVL    28(AP), PRIMPTR                                 ; 6870
                                                 39  11 0002E            BRB     4$                                              ; 6840
                                                 50  D5 00030 2$:        TSTL    R0                                              ; 6882
                                                 0C  13 00032            BEQL    3$                                              ;
                                58               50  D0 00034            MOVL    R0, TOKEN                                       ; 6885
                                   00000000'     EF  D4 00037            CLRL    SAVED_TOKEN                                     ; 6886
                                         00D1    31  00 0003D            BRW     9$                                              ; 6887
                                                 56  D4 00040 3$:        CLRL    LAST_OPERAND                                    ; 6897
                                10   AE      04   AC  D0 00042            MOVL    OPERAND_EXPECTED_FLAG, OPERAND_EXPECTED         ; 6898
                                                 59  D4 00047            CLRL    STATE_INDEX                                     ; 6899
                                            24   AE  D4 00049            CLRL    PRIMPTR                                         ; 6900
                                            08   AE  D4 0004C            CLRL    NUMERIC_PATHNAME                                ; 6901
0DD0    8F          00           6E              00  2C 0004F            MOVC5   #0, (SP), #0, #208, PATHDESC                    ; 6902
                                          FF30   CD      00056                                                                   ;
                                5B        FF38   CD  9E 00059            MOVAB   PATHDESC+8, PATHVECTOR                          ; 6903
0270    8F          00           6E              00  2C 0005E            MOVC5   #0, (SP), #0, #624, SUBSCR_DESC                 ; 6904
                                            40   AE      00065                                                                   ;
                                                 57  D4 00067            CLRL    PLIPTR                                          ; 6905
                                7E           0C  AC  7D 00069 4$:        MOVQ    TERM_LIST, -(SP)                                ; 6919
                                            08   AC  DD 0006D            PUSHL   ADDRESS_EXPRESSION                              ;
                                            1C   AE  DD 00070            PUSHL   OPERAND_EXPECTED                                ; 6918
                                      F098       CF  04  FB 00073            CALLS   #4, DBG$LEXICAL_SCANNER                         ;
                                                 58  D0 00078            MOVL    R0, TOKEN                                       ;
                                07 00000000G     00      02  E1 0007B            BBC     #2, DBG$GL_DEVELOPER, 5$                ; 6920
                                                 58  DD 00083            PUSHL   TOKEN                                           ;
                                      0000V      CF  01  FB 00085            CALLS   #1, DUMP_TOKEN                                  ;
                                                 5A           24  AE  D0 0008A 5$:        MOVL    PRIMPTR, R10                    ; 6929
                                                 55      12  00 0008E            BNEQ    6$                                      ;
                                                 51           10  AE  E8 00090            BLBS    OPERAND_EXPECTED, 6$            ; 6930
                                                 56  D5 00094            TSTL    LAST_OPERAND                                    ; 6931
                                                 4D  13 00096            BEQL    6$                                              ;
                                                 01          68  91 00098            CMPB    (TOKEN), #1                             ; 6932
                                                 48  12 0009B            BNEQ    6$                                              ;
                                                 50           02  A8 3C 0009D            MOVZWL  2(TOKEN), R0                    ; 6933
                                3C 00000000'     EF      50  E1 000A1            BBC     R0, TOKEN_IS_INTEGER, 6$                ;
                                                 50           08  A8 9A 000A9            MOVZBL  8(TOKEN), R0                    ; 6937
                                                 50  C0 000AD            ADDL2   #4, R0                                          ;
                                                 50  C6 000B0            DIVL2   #4, R0                                          ;
                                                 03  A0 9F 000B3            PUSHAB  3(R0)                                           ; 6936
                                   00000000G     00      01  FB 000B6            CALLS   #1, DBG$GET_TEMPMEM                             ;
                                         04      AE      50  D0 000BD            MOVL    R0, TEMPTOKEN                           ;
                                         04      BE      04  90 000C1            MOVB    #4, @TEMPTOKEN                          ; 69[illegible]
                                         04      BE  0100    8F A8 000C5            BISW2   #256, @TEMPTOKEN                        ; 6939
04      BE          10                   10              09  F0 000CB            INSV    #9, #16, #16, @TEMPTOKFN                ; 6940
                                                 50           08  A8 9A 000D1            MOVZBL  8(TOKEN), R0                    ; 6941
                                                 50  D6 000D5            INCL    R0                                              ;
                                7E       04      AE          0C  C1 000D7            ADDL3   #12, TEMPTOKEN, -(SP)                   ; 6942
                                9E       08      A8          50  28 000DC            MOVC3   R0, 8(TOKEN), @(SP)+                    ;
```

```
                58        04    AE  D0 000E1          MOVL    TEMPTOKEN, TOKEN                        ; 6943
                01              68  91 000E5 6$:       CMPB    (TOKEN), #1                             ; 6951
                                1F  12 000E8           BNEQ    8$
                12        10    AE  E8 000EA           BLBS    OPERAND_EXPECTED, 7$                    ; 6954
                          08    A8  9F 000EE           PUSHAB  8(TOKEN)                                ; 6956
                                01  DD 000F1           PUSHL   #1
                    000289AA    8F  DD 000F3           PUSHL   #166314
00000000G       00              03  FB 000F9           CALLS   #3, LIB$SIGNAL
                          10    AE  D4 00100 7$:       CLRL    OPERAND_EXPECTED                        ; 6958
                56              58  D0 00103           MOVL    TOKEN, LAST_OPERAND                     ; 6959
                              FF60  31 00106           BRW     4$                                      ; 6951
                1E        01    A8  E8 00109 8$:       BLBS    1(TOK, ', 11$                           ; 6979
                                56  D5 0010D           TSTL    LAST_OPERAND                            ; 6986
                                08  12 0010F           BNEQ    10$
             14 BC              58  D0 00111 9$:       MOVL    TOKEN, @RET_TOKEN                       ; 6989
                          18    BC  D4 00115           CLRL    @RET_OPERAND_FLAG                       ; 6990
                                04  00118             RET                                             ; 6988
00000000'       EF              58  D0 00119 10$:      MOVL    TOKEN, SAVED_TOKEN                      ; 6998
                58 00000000'    EF  9E 00120           MOVAB   PRIMARY_TERM_TOKEN, TOKEN               ; 6999
             18 BC              01  D0 00127           MOVL    #1, @RET_OPERAND_FLAG                   ; 7000
                09        10    AE  E9 0012B 11$:      BLBC    OPERAND_EXPECTED, 12$                   ; 7010
                02              68  91 0012F           CMPB    (TOKEN), #2                             ; 7011
                                09  12 00132           BNEQ    13$
                17        10    AE  E8 00134           BLBS    OPERAND_EXPECTED, 14$                   ; 7012
                02              68  91 00138 12$:      CMPB    (TOKEN), #2                             ; 7013
                                12  12 0013B           BNEQ    14$
                          0C    A8  9F 0013D 13$:      PUSHAB  12(TOKEN)                               ; 7015
                                01  DD 00140           PUSHL   #1
                    000289B2    8F  DD 00142           PUSHL   #166322
00000000G       00              03  FB 00148           CALLS   #3, LIB$SIGNAL
                02              68  91 0014F 14$:      CMPB    (TOKEN), #2                             ; 7018
                                24  13 00152           BEQL    16$
                                56  D5 00154           TSTL    LAST_OPERAND                            ; 7020
                                0E  13 00156           BEQL    15$
                FFFFFFFF        8F  56  D1 00158       CMPL    LAST_OPERAND, #-1                       ; 7025
                                17  13 0015F           BEQL    16$
                01              66  91 00161           CMPB    (LAST_OPERAND), #1                      ; 7028
                                12  13 00164           BEQL    16$
                          0C    A8  9F 00166 15$:      PUSHAB  12(TOKEN)                               ; 7030
                                01  DD 00169           PUSHL   #1
                    000289B2    8F  DD 0016B           PUSHL   #166322
                00000000G       00  03 FB 00171       CALLS   #3, LIB$SIGNAL
                04              68  91 00178 16$:      CMPB    (TOKEN), #4                             ; 7035
                                04  13 0017B           BEQL    17$
                      10        AE  01 D0 0017D        MOVL    #1, OPERAND_EXPECTED                    ; 7037
                      14        AE  02 A8 3C 00181 17$: MOVZWL  2(TOKEN), OPCODE                        ; 7048
                    00000000'FF49  DF  00186 18$:      PUSHAL  @PRIMARY_TABLE[STATE_INDEX]             ; 7049
14    AE          9E        08        00  ED 0018D     CMPZV   #0, #8, @(SP)+, OPCODE
                                21  13 00193           BEQL    20$
                    00000000'FF49  DF  00195           PUSHAL  @PRIMARY_TABLE[STATE_INDEX]             ; 7051
                                9E  95 0019C           TSTB    @(SP)+
                                12  12 0019E           BNEQ    19$
                          0C    A8  9F 001A0           PUSHAB  12(TOKEN)                               ; 7053
                                01  DD 001A3           PUSHL   #1
                    000289E2    8F  DD 001A5           PUSHL   #166370
                00000000G       00  03 FB 001AB       CALLS   #3, LIB$SIGNAL
                                59  D6 001B2 19$:      INCL    STATE_INDEX                             ; 7055
```

```
                                            D0  11 001B4        BRB     18$                              ; 7049
                                 00000000'FF49  DF 001B6 20$:   PUSHAL  @PRIMARY_TABLE[STATE_INDEX]      ; 7058
  18    AE          9E          08          08  EF 001BD        EXTZV   #8, #8, @(SP)+, ACTION           ;
                                 00000000'FF49  DF 001C3        PUSHAL  @PRIMARY_TABLE[STATE_INDEX]      ; 7059
        59          9E          10          10  EF 001CA        EXTZV   #16, #16, @(SP)+, STATE_INDEX    ;
                    31          01      18  AE  CF 001CF        CASEL   ACTION, #1, #49                  ; 7064
      0167        008C        007B        FE95     001D4 21$:   .WORD   4$-21$,-                         ;
      01F9        01B0        057D        0569     001DC                22$-21$,-                        ;
      0402        029D        0254        059E     001E4                23$-21$,-                        ;
      047B        00F3        0462        02E2     001EC                31$-21$,-                        ;
      04CB        059E        04A0        0569     001F4                84$-21$,-                        ;
      0569        0448        0524        04FB     001FC                88$-21$,-                        ;
      058D        057D        0579        056D     00204                36$-21$,-                        ;
      05BB        05AE        059E        059A     0020C                40$-21$,-                        ;
      063F        0448        063A        062D     00214                93$-21$,-                        ;
      0656        0648        06E2        FE95     0021C                44$-21$,-                        ;
      FE95        06CA        06C1        0660     00224                48$-21$,-                        ;
      0352        06F5        06EB        06E2     0022C                66$-21$,-                        ;
                              0600        0389     00234                50$-21$,-                        ;
                                                                        72$-21$,-                        ;
                                                                        27$-21$,-                        ;
                                                                        74$-21$,-                        ;
                                                                        84$-21$,-                        ;
                                                                        75$-21$,-                        ;
                                                                        93$-21$,-                        ;
                                                                        77$-21$,-                        ;
                                                                        80$-21$,-                        ;
                                                                        82$-21$,-                        ;
                                                                        69$-21$,-                        ;
                                                                        84$-21$,-                        ;
                                                                        85$-21$,-                        ;
                                                                        87$-21$,-                        ;
                                                                        88$-21$,-                        ;
                                                                        91$-21$,-                        ;
                                                                        92$-21$,-                        ;
                                                                        93$-21$,-                        ;
                                                                        95$-21$,-                        ;
                                                                        96$-21$,-                        ;
                                                                        103$-21$,-                       ;
                                                                        105$-21$,-                       ;
                                                                        69$-21$,-                        ;
                                                                        106$-21$,-                       ;
                                                                        4$-21$,-                         ;
                                                                        117$-21$,-                       ;
                                                                        107$-21$,-                       ;
                                                                        108$-21$,-                       ;
                                                                        109$-21$,-                       ;
                                                                        114$-21$,-                       ;
                                                                        115$-21$,-                       ;
                                                                        4$-21$,-                         ;
                                                                        117$-21$,-                       ;
                                                                        119$-21$,-                       ;
                                                                        122$-21$,-                       ;
                                                                        55$-21$,-                        ;
                                                                        58$-21$,-                        ;
                                                                        98$-21$                          ;
                                  00000000'  EF  9F 00238       PUSHAB  P.AXK                            ; 8247
```

```
                                01  DD 0023E          PUSHL   #1                                          ;
                        00028362    8F  DD 00240          PUSHL   #164706                                     ;
                00000000G   00      03  FB 00246          CALLS   #3, LIB$SIGNAL                              ;
                                    76  11 0024D          BRB     26$                                         ;
                    FF30    CD  0101    8F  B0 0024F 22$:  MOVW    #257, PATHDESC                              ; 7081
                        6B 00000000'    EF  9E 00256       MOVAB   P.AXG, (PATHVECTOR)                         ; 7083
                                  03BC  31  00025D         BRW     69$                                         ; 7084
                        50      02      A6  3C 00260 23$:  MOVZWL  2(LAST_OPERAND), R0                         ; 7106
            03 00000000'    EF          50  E0 00264       BBS     R0, TOREN_IS_INTEGER, 24$                   ;
                                  04CE  31  00026C         BRW     84$                                         ;
                    FE5A    CD  010E    8F  B0 0026F 24$:  MOVW    #270, STRDESC+2                             ; 7109
                    FE58    CD    08    A6  9B 00276       MOVZBW  8(LAST_OPERAND), STRDESC                    ; 7111
                    FE5C    CD    09    A6  9E 0027C       MOVAB   9(R6), STRDESC+4                            ; 7112
                            FF34  CD  9F  00282            PUSHAB  PATHDESC+4                                  ; 7113
                            FE58  CD  9F  00286            PUSHAB  STRDESC                                     ;
                00000000G   00          02  FB 0028A       CALLS   #2, OTS$CVT_TI_L                            ;
                       0C   AE          50  D0 00291       MOVL    R0, STATUS                                  ;
                        15      0C      AE  E8 00295       BLBS    STATUS, 25$                                 ; 7114
                                0C      AE  DD 00299       PUSHL   STATUS                                      ; 7117
                                08      A6  9F 0029C       PUSHAB  8(LAST_OPERAND)                             ;
                                    01  DD 0029F          PUSHL   #1                                          ;
                        0002898A    8F  DD 002A1           PUSHL   #166282                                     ;
                00000000G   00          04  FB 002A7       CALLS   #4, LIB$SIGNAL                              ;
                    FF30    CD          01  90 002AE 25$:  MOVB    #1, PATHDESC                                ; 7120
                    FF31    CD  0101    8F  B0 002B3       MOVW    #257, PATHDESC+1                            ; 7121
                        6B 00000000'    EF  9E 002BA       MOVAB   P.AXH, (PATHVECTOR)                         ; 7122
                       08   AE          01  D0 002C1       MOVL    #1, NUMERIC_PATHNAME                        ; 7123
                                    72  11 002C5 26$:  BRB     30$                                         ; 7106
                            FF32    CD  95  002C7 27$:  TSTB    PATHDESC+2                                  ; 7149
                                    0D  13  002CB       BEQL    28$                                         ;
                        00028A12    8F  DD 002CD           PUSHL   #166418                                     ; 7151
                00000000G   00          01  FB 002D3       CALLS   #1, LIB$SIGNAL                              ;
                    FE5A    CD  010E    8F  B0 002DA 28$:  MOVW    #270, STRDESC+2                             ; 7157
                    FE58    CD    0C    A8  9B 002E1       MOVZBW  12(TOKEN), STRDESC                          ; 7159
                    FE5C    CD    0D    A8  9E 002E7       MOVAB   13(R8), STRDESC+4                           ; 7160
                            FF34  CD  9F  002ED            PUSHAB  PATHDESC+4                                  ; 7161
                            FE58  CD  9F  002F1            PUSHAB  STRDESC                                     ;
                00000000G   00          02  FB 002F5       CALLS   #2, OTS$CVT_TI_L                            ;
                       0C   AE          50  D0 002FC       MOVL    R0, STATUS                                  ;
                        2D      0C      AE  E8 00300       BLBS    STATUS, 29$                                 ; 7162
                                0C      AE  DD 00304       PUSHL   STATUS                                      ; 7167
                            00000000'   EF  9F 00307       PUSHAB  P.AXJ                                       ; 7166
                                FE58    CD  DD 0030D       PUSHL   STRDESC                                     ;
                            00000000'   EF  9F 00311       PUSHAB  P.AXI                                       ; 7165
                                    03  DD 00317          PUSHL   #3                                          ; 7164
                            00028E78    8F  DD 00319       PUSHL   #167544                                     ;
                                  0C    A8  9F 0031F       PUSHAB  12(TOKEN)                                   ;
                                    01  DD 00322          PUSHL   #1                                          ;
                            0002898A    8F  DD 00324       PUSHL   #166282                                     ;
                00000000G   00          09  FB 0032A       CALLS   #9, LIB$SIGNAL                              ;
FF32    CD      FF30    CD              01  81 00331 29$:  ADDB3   #1, PATHDESC, PATHDESC+2                    ; 7169
                                        46  11 00339 30$:  BRB     35$                                         ; 7064
                        01      02      A6  B1 0033B 31$:  CMPW    2(LAST_OPERAND), #1                         ; 7186
                                    1F  12  0033F          BNEQ    32$                                         ;
                                FF32    CD  95  00341      TSTB    PATHDESC+2                                  ; 7194
                                    19  12  00345          BNEQ    32$                                         ;
                                  20    AE  9F  00347      PUSHAB  DUMMY                                       ; 7201
```

```
                        24   AE   9F 0034A          PUSHAB  DUMMY                                   ;
                        2C   AE   9F 0034D          PUSHAB  PRIMPTR                                 ;
                        3C   AE   9F 00350          PUSHAB  KIND                                    ;
                        08   A6   9F 00353          PUSHAB  8(LAST_OPERAND)                         ;
00000000G               00        05 FB 00356       CALLS   #5, DBG$DEF_SYM_FIND                    ;
                        03        50 E8 0035D       BLBS    R0, 33$                                 ;
                             02EC 31 00360 32$:     BRW     74$                                     ;
                        01   30   AE D1 00363 33$:  CMPL    KIND, #1                                ; 7206
                             06   13 00367         BEQL    34$                                     ;
                        05   30   AE D1 00369       CMPL    KIND, #5                                ; 7207
                             F1   12 0036D         BNEQ    32$                                     ;
                             7E   D4 0036F 34$:    CLRL    -(SP)                                   ; 7216
                        24   AE   9F 00371         PUSHAB  DUMMY                                   ;
                        2C   AE   9F 00374         PUSHAB  PRIMPTR                                 ;
                        30   AE   DD 00377         PUSHL   PRIMPTR                                 ;
00000000G               00        04 FB 0037A       CALLS   #4, DBG$NCOPY_DESC                      ;
                             FCE5 31 00381 35$:     BRW     4$                                      ; 7218
                        01   02   A6 B1 00384 36$:  CMPW    2(LAST_OPERAND), #1                     ; 7263
                             1F   12 00388         BNEQ    37$                                     ;
                        FF32 CD   95 0038A         TSTB    PATHDESC+2                              ; 7271
                             19   12 0038E         BNEQ    37$                                     ;
                        20   AE   9F 00390         PUSHAB  DUMMY                                   ; 7278
                        24   AE   9F 00393         PUSHAB  DUMMY                                   ;
                        2C   AE   9F 00396         PUSHAB  PRIMPTR                                 ;
                        3C   AE   9F 00399         PUSHAB  KIND                                    ;
                        08   A6   9F 0039C         PUSHAB  8(LAST_OPERAND)                         ;
00000000G               00        05 FB 0039F       CALLS   #5, DBG$DEF_SYM_FIND                    ;
                        03        50 E8 003A6       BLBS    R0, 38$                                 ;
                             02C8 31 003A9 37$:     BRW     75$                                     ;
                        01   30   AE D1 003AC 38$:  CMPL    KIND, #1                                ; 7283
                             06   13 003B0         BEQL    39$                                     ;
                        05   30   AE D1 003B2       CMPL    KIND, #5                                ; 7284
                             F1   12 003B6         BNEQ    37$                                     ;
                             7E   D4 003B8 39$:    CLRL    -(SP)                                   ; 7293
                        24   AE   9F 003BA         PUSHAB  DUMMY                                   ;
                        2C   AE   9F 003BD         PUSHAB  PRIMPTR                                 ;
                        30   AE   DD 003C0         PUSHL   PRIMPTR                                 ;
00000000G               00        04 FB 003C3       CALLS   #4, DBG$NCOPY_DESC                      ;
                             02CC 31 003CA         BRW     76$                                     ; 7305
                        01   02   A6 B1 003CD 40$:  CMPW    2(LAST_OPERAND), #1                     ; 7333
                             40   12 003D1         BNEQ    42$                                     ;
                        FF32 CD   95 003D3         TSTB    PATHDESC+2                              ; 7341
                             4C   12 003D7         BNEQ    43$                                     ;
                        20   AE   9F 003D9         PUSHAB  DUMMY                                   ; 7348
                        24   AE   9F 003DC         PUSHAB  DUMMY                                   ;
                        2C   AE   9F 003DF         PUSHAB  PRIMPTR                                 ;
                        3C   AE   9F 003E2         PUSHAB  KIND                                    ;
                        08   A6   9F 003E5         PUSHAB  8(LAST_OPERAND)                         ;
00000000G               00        05 FB 003E8       CALLS   #5, DBG$DEF_SYM_FIND                    ;
                        33        50 E9 003EF       BLBC    R0, 43$                                 ;
                        01   30   AE D1 003F2       CMPL    KIND, #1                                ; 7353
                             06   13 003F6         BEQL    41$                                     ;
                        05   30   AE D1 003F8       CMPL    KIND, #5                                ; 7354
                             27   12 003FC         BNEQ    43$                                     ;
                             7E   D4 003FE 41$:    CLRL    -(SP)                                   ; 7363
                        24   AE   9F 00400         PUSHAB  DUMMY                                   ;
                        2C   AE   9F 00403         PUSHAB  PRIMPTR                                 ;
```

```
                        30   AE   DD 00406          PUSHL   PRIMPTR                                    ;
00000000G   00               04   FB 00409          CALLS   #4, DBG$NCOPY_DESC                        ;
                         02AF   31 00410          BRW     78$                                       ; 7365
                        08   A6   9F 00413 42$:     PUSHAB  8(LAST_OPERAND)                           ; 7376
                        01   DD   0C416              PUSHL   #1                                        ;
                    000281A8   8F   DD 00418          PUSHL   #164264                                   ;
00000000G   00               03   FB 0041E          CALLS   #3, LIB$SIGNAL                            ;
                         0277   31 00425 43$:     BRW     77$                                       ; 7382
                01       02   A6   B1 00428 44$:     CMPW    2(LAST_OPERAND), #1                       ; 7407
                             1F   12 0042C          BNEQ    45$                                       ;
                          FF32   CD   95 0042E          TSTB    PATHDESC+2                                ; 7415
                             19   12 00432          BNEQ    45$                                       ;
                        20   AE   9F 00434          PUSHAB  DUMMY                                     ; 7422
                        24   AE   9F 00437          PUSHAB  DUMMY                                     ;
                        2C   AE   9F 0043A          PUSHAB  PRIMPTR                                   ;
                        3C   AE   9F 0043D          PUSHAB  KIND                                      ;
                        08   A6   9F 00440          PUSHAB  8(LAST_OPERAND)                           ;
00000000G   00               05   FB 00443          CALLS   #5, DBG$DEF_SYM_FIND                      ;
                03               50   E8 0044A          BLBS    R0, 46$                                   ;
                         027F   31 0044D 45$:     BRW     80$                                       ;
                01       30   AE   D1 00450 46$:     CMPL    KIND, #1                                  ; 7427
                             06   13 00454          BEQL    47$                                       ;
                05       30   AE   D1 00456          CMPL    KIND, #5                                  ; 7428
                             F1   12 0045A          BNEQ    45$                                       ;
                             7E   D4 0045C 47$:     CLRL    -(SP)                                     ; 7437
                        24   AE   9F 0045E          PUSHAB  DUMMY                                     ;
                        2C   AE   9F 00461          PUSHAB  PRIMPTR                                   ;
                        30   AE   DD 00464          PUSHL   PRIMPTR                                   ;
00000000G   00               04   FB 00467          CALLS   #4, DBG$NCOPY_DESC                        ;
                         0281   31 0046E          BRW     81$                                       ; 7449
                             7E   D4 00471 48$:     CLRL    -(SP)                                     ; 7465
                             56   DD 00473          PUSHL   LAST_OPERAND                              ;
                          FF30   CD   9F 00475          PUSHAB  PATHDESC                                  ;
0000V   CF                   03   FB 00479          CALLS   #3, APPEND_TO_PATHNAME                    ;
                          FE60   CD   9F 0047E          PUSHAB  SAVED_PATHDESC                            ; 7466
                             57   DD 00482          PUSHL   PLIPTR                                    ; 7467
                        48   AE   9F 00484          PUSHAB  SUBSCR_DESC                               ; 7466
                          FF30   CD   9F 00487          PUSHAB  PATHDESC                                  ;
0000V   CF                   04   FB 0048B          CALLS   #4, PATHNAME_TO_PRIMARY                   ;
                57               50   D0 00490          MOVL    R0, PLIPTR                                ;
                06       07   A7   91 00493          CMPB    @(PLIPTR), #6                             ; 7468
                             09   12 00497          BNEQ    49$                                       ;
                10       06   A7   91 00499          CMPB    6(PLIPTR), #16                            ; 7469
                             03   12 0049D          BNEQ    49$                                       ;
                         03D3   31 0049F          BRW     112$                                      ;
                        1C   AE   9F 004A2 49$:     PUSHAB  NAME                                      ; 7474
                          FF30   CD   9F 004A5          PUSHAB  PATHDESC                                  ;
00000000G   00               02   FB 004A9          CALLS   #2, DBG$NPATHDESC_TO_CS                   ;
                        1C   AE   DD 004B0          PUSHL   NAME                                      ; 7475
                         03B0   31 004B3          BRW     111$                                      ;
                10       02   A6   B1 004B6 50$:     CMPW    2(LAST_OPERAND), #16                      ; 7499
                             12   13 004BA          BEQL    51$                                       ;
                        0C   A8   9F 004BC          PUSHAB  12(TOKEN)                                 ; 7501
                             01   DD 004BF          PUSHL   #1                                        ;
                    000289B2   8F   DD 004C1          PUSHL   #166322                                   ;
00000000G   00               03   FB 004C7          CALLS   #3, LIB$SIGNAL                            ;
                        08   A6   9F 004CE 51$:     PUSHAB  8(LAST_OPERAND)                           ; 7510
```

```
                        7E      01    A6  9A 004D1          MOVZBL  1(LAST_OPERAND), -(SP)                ;
                  EBAA  CF            02  FB 004D5          CALLS   #2, DBG$GET_BIF_ARGUMENTS            ;
                        6E            50  D0 004DA          MOVL    R0, ARG_LIST                         ;
                        02      00    BE  D1 004DD          CMPL    @ARG_LIST, #2                        ; 7516
                                      35  14 004E1          BGTR    54$                                  ;
                        01      00    BE  D1 004E3          CMPL    @ARG_LIST, #1                        ; 7525
                                      05  12 004E7          BNEQ    52$                                  ;
                        7E            68  9A 004E9          MOVZBL  (TOKEN), -(SP)                       ; 7529
                                      02  11 004EC          BRB     53$                                  ; 7528
                                      03  DD 004EE 52$:      PUSHL   #3                                   ; 7532
                              08      A6  9F 004F0 53$:      PUSHAB  8(LAST_OPERAND)                      ;
                        7E    04      A6  9A 004F3          MOVZBL  4(LAST_OPERAND), -(SP)               ;
                  0000V CF            03  FB 004F7          CALLS   #3, CREATE_OPERATOR_TOKEN            ;
                        58            50  D0 004FC          MOVL    R0, TOKEN                            ;
                52      6E            08  C1 004FF          ADDL3   #8, ARG_LIST, R2                     ; 7535
                                      62  DD 00503          PUSHL   (R2)                                 ;
                53      04    AE      04  C1 00505          ADDL3   #4, ARG_LIST, R3                     ;
                                      63  DD 0050A          PUSHL   (R3)                                 ;
                                      58  DD 0050C          PUSHL   TOKEN                                ;
            00000000G   00            03  FB 0050E          CALLS   #3, DBG$EVAL_LANG_OPERATOR           ;
                                  02DF  31 00515          BRW     102$                                 ;
                              08      A6  9F 00518 54$:      PUSHAB  8(LAST_OPERAND)                      ; 7546
                                      01  DD 0051B          PUSHL   #1                                   ;
                          00028838    8F  DD 0051D          PUSHL   #165944                              ;
                                  00A7  31 00523          BRW     65$                                  ;
                        01        02    A6  B1 00526 55$:      CMPW    2(LAST_OPERAND), #1                  ; 7563
                                      03  13 0052A          BEQL    56$                                  ;
                                  0092  31 0052C          BRW     64$                                  ;
                              FF32    CD  95 0052F 56$:      TSTB    PATHDESC+2                           ; 7571
                                      28  12 00533          BNEQ    58$                                  ;
                                20    AE  9F 00535          PUSHAB  DUMMY                                ; 7578
                                24    AE  9F 00538          PUSHAB  DUMMY                                ;
                                2C    AE  9F 0053B          PUSHAB  PRIMPTR                              ;
                                3C    AE  9F 0053E          PUSHAB  KIND                                 ;
                                08    A6  9F 00541          PUSHAB  8(LAST_OPERAND)                      ;
            00000000G   00            05  FB 00544          CALLS   #5, DBG$DEF_SYM_FIND                 ;
                        0F            50  E9 0054B          BLBC    R0, 58$                              ;
                        01      30    AE  D1 0054E          CMPL    KIND, #1                             ; 7583
                                      04  13 00552          BEQL    57$                                  ;
                        05      30    AE  D1 00554          CMPL    KIND, #5                             ; 7584
                                      03  12 00558 57$:      BNEQ    58$                                  ;
                                  00AA  31 0055A          BRW     67$                                  ;
                                      7E  D4 0055D 58$:      CLRL    -(SP)                                ; 7602
                                      56  DD 0055F 59$:      PUSHL   LAST_OPERAND                         ;
                              FF30    CD  9F 00561          PUSHAB  PATHDESC                             ;
                  0000V CF            03  FB 00565          CALLS   #3, APPEND_TO_PATHNAME               ;
                                      01  DD 0056A          PUSHL   #1                                   ; 7608
                                      7E  7C 0056C          CLRQ    -(SP)                                ;
                                      7E  D4 0056E          CLRL    -(SP)                                ;
                                40    AE  9F 00570          PUSHAB  KIND                                 ;
                                48    AE  9F 00573          PUSHAB  TYPEID                               ;
                              FF30    CD  9F 00576          PUSHAB  PATHDESC                             ;
            00000000G   00            07  FB 0057A          CALLS   #7, DBG$STA_GETSYMBOL                ;
                                34    AE  D5 00581          TSTL    TYPEID                               ; 7613
                                      03  13 00584          BEQL    60$                                  ;
                                  0262  31 00586          BRW     101$                                 ;
                                38    AE  9F 00589 60$:      PUSHAB  PATHSTRING                           ; 7616
```

```
                        FF30   CD  9F 0058C          PUSHAB  PATHDESC                                        ;
00000000G   00                 02  FB 00590          CALLS   #2, DBG$NPATHDESC_TO_CS                        ;
            09           30    AE  D1 00597          CMPL    KIND, #9                                       ; 7617
                             0D  12 0059B          BNEQ    61$                                            ;
                           38    AE  DD 0059D          PUSHL   PATHSTRING                                     ; 7619
                           01    DD  005A0          PUSHL   #1                                             ;
                      000281F0   8F  DD 005A2          PUSHL   #164336                                        ;
                             14  11  005A8          BRB     63$                                            ;
            0D           30    AE  D1 005AA 61$:     CMPL    KIND, #13                                      ; 7621
                             03  13  005AE          BEQL    62$                                            ;
                           0226  31  005B0          BRW     99$                                            ;
                           38    AE  DD 005B3 62$:     PUSHL   PATHSTRING                                     ; 7623
                           01    DD  005B6          PUSHL   #1                                             ;
                      000282A8   8F  DD 005B8          PUSHL   #164520                                        ;
                           0223  31  005BE 63$:     BRW     100$                                           ; 7625
            7E           08    A6  9A 005C1 64$:     MOVZBL  8(LAST_OPERAND), -(SP)                         ; 7640
                           01    DD  005C5          PUSHL   #1                                             ;
                      000289E2   8F  DD 005C7          PUSHL   #166370                                        ;
00000000G   00                 03  FB 005CD 65$:     CALLS   #3, LIB$SIGNAL                                 ;
                             43  11  005D4          BRB     68$                                            ; 7556
            01           02    A6  B1 005D6 66$:     CMPW    2(LAST_OPERAND), #1                            ; 7658
                             50  12  005DA          BNEQ    71$                                            ;
                        FF32   CD  95 005DC          TSTB    PATHDESC+2                                     ; 7666
                             3A  12  005E0          BNEQ    69$                                            ;
                           20    AE  9F 005E2          PUSHAB  DUMMY                                          ; 7673
                           24    AE  9F 005E5          PUSHAB  DUMMY                                          ;
                           2C    AE  9F 005E8          PUSHAB  PRIMPTR                                        ;
                           3C    AE  9F 005EB          PUSHAB  KIND                                           ;
                           08    A6  9F 005EE          PUSHAB  8(LAST_OPERAND)                                ;
00000000G   00                 05  FB 005F1          CALLS   #5, DBG$DEF_SYM_FIND                           ;
            21                 50  E9 005F8          BLBC    R0, 69$                                        ;
            01           30    AE  D1 005FB          CMPL    KIND, #1                                       ; 7678
                             06  13  005FF          BEQL    67$                                            ;
            05           30    AE  D1 00601          CMPL    KIND, #5                                       ; 7679
                             15  12  00605          BNEQ    69$                                            ;
                             7E  D4  00607 67$:     CLRL    -(SP)                                          ; 7688
                           24    AE  9F 00609          PUSHAB  DUMMY                                          ;
                           2C    AE  9F 0060C          PUSHAB  PRIMPTR                                        ;
                           30    AE  DD 0060F          PUSHL   PRIMPTR                                        ;
00000000G   00                 04  FB 00612          CALLS   #4, DBG$NCOPY_DESC                             ;
                         02AD  31  00619 68$:     BRW     122$                                           ; 7681
                             7E  D4  0061C 69$:     CLRL    -(SP)                                          ; 7697
                             56  DD  0061E 70$:     PUSHL   LAST_OPERAND                                   ;
                        FF30   CD  9F 00620          PUSHAB  PATHDESC                                       ;
      0000V CF                 03  FB 00624          CALLS   #3, APPEND_TO_PATHNAME                         ;
                         0272  31  00629          BRW     115$                                           ; 7698
                             56  DD  0062C 71$:     PUSHL   LAST_OPERAND                                   ; 7707
      0000V CF                 01  FB 0062E          CALLS   #1, CONSTANT_TO_VALDESCR                       ;
                         027A  31  00633          BRW     116$                                           ;
            12           08    AE  E9 00636 72$:     BLBC    NUMERIC_PATHNAME, 73$                          ; 7724
                           08    A6  9F 0063A          PUSHAB  8(LAST_OPERAND)                                ; 7726
                           01    DD  0063D          PUSHL   #1                                             ;
                  0002898A   8F  DD 0063F          PUSHL   #166282                                        ;
00000000G   00                 03  FB 00645          CALLS   #3, LIB$SIGNAL                                 ;
                         00EE  31  0064C 73$:     BRW     84$                                            ; 7728
                             7E  D4  0064F 74$:     CLRL    -(SP)                                          ; 7738
                             56  DD  00651          PUSHL   LAST_OPERAND                                   ;
```

```
                FF30    CD  9F 00653          PUSHAB  PATHDESC                        ;
0000V   CF              03  FB 00657          CALLS   #3, APPEND_TO_PATHNAME          ;
                FE60    CD  9F 0065C          PUSHAB  SAVED_PATHDESC                  ; 7739
                        57  DD 00660          PUSHL   PLIPTR                          ; 7740
                  48    AE  9F 00662          PUSHAB  SUBSCR_DESC                     ; 7739
                FF30    CD  9F 00665          PUSHAB  PATHDESC                        ;
0000V   CF              04  FB 00669          CALLS   #4, PATHNAME_TO_PRIMARY         ;
  24    AE              50  D0 0066E          MOVL    R0, PRIMPTR                     ;
                        59  11 00672          BRB     79$                             ; 7064
                        7E  D4 00674 75$:     CLRL    -(SP)                           ; 7761
                        56  DD 00676          PUSHL   LAST_OPERAND                    ;
                FF30    CD  9F 00678          PUSHAB  PATHDESC                        ;
0000V   CF              03  FB 0067C          CALLS   #3, APPEND_TO_PATHNAME          ;
                        01  DD 00681          PUSHL   #1                              ; 7762
                FE60    CD  9F 00683          PUSHAB  SAVED_PATHDESC                  ;
                        57  DD 00687          PUSHL   PLIPTR                          ; 7763
                  4C    AE  9F 00689          PUSHAB  SUBSCR_DESC                     ; 7762
                FF30    CD  9F 0068C          PUSHAB  PATHDESC                        ;
0000V   CF              05  FB 00690          CALLS   #5, PATHNAME_TO_PRIMARY         ;
  24    AE              50  D0 00695          MOVL    R0, PRIMPTR                     ;
                  24    AE  DD 00699 76$:     PUSHL   PRIMPTR                         ; 7764
                      0219  31 0069C          BRW     118$                            ;
                        7E  D4 0069F 77$:     CLRL    -(SP)                           ; 7787
                        56  DD 006A1          PUSHL   LAST_OPERAND                    ;
                FF30    CD  9F 006A3          PUSHAB  PATHDESC                        ;
0000V   CF              03  FB 006A7          CALLS   #3, APPEND_TO_PATHNAME          ;
                FE60    CD  9F 006AC          PUSHAB  SAVED_PATHDESC                  ; 7788
                        57  DD 006B0          PUSHL   PLIPTR                          ; 7789
                  48    AE  9F 006B2          PUSHAB  SUBSCR_DESC                     ; 7788
                FF30    CD  9F 006B5          PUSHAB  PATHDESC                        ;
0000V   CF              04  FB 006B9          CALLS   #4, PATHNAME_TO_PRIMARY         ;
  24    AE              50  D0 006BE          MOVL    R0, PRIMPTR                     ;
                  08    A6  9F 006C2 78$:     PUSHAB  8(LAST_OPERAND)                 ; 7790
                  28    AE  DD 006C5          PUSHL   PRIMPTR                         ;
0000V   CF              02  FB 006C8          CALLS   #2, GET_BLISS_SUBSCRIPTS        ;
                        7C  11 006CD 79$:     BRB     86$                             ; 7064
                        7E  D4 006CF 80$:     CLRL    -(SP)                           ; 7803
                        56  DD 006D1          PUSHL   LAST_OPERAND                    ;
                FF30    CD  9F 006D3          PUSHAB  PATHDESC                        ;
0000V   CF              03  FB 006D7          CALLS   #3, APPEND_TO_PATHNAME          ;
                FE60    CD  9F 006DC          PUSHAB  SAVED_PATHDESC                  ; 7804
                        57  DD 006E0          PUSHL   PLIPTR                          ; 7805
                  48    AE  9F 006E2          PUSHAB  SUBSCR_DESC                     ; 7804
                FF30    CD  9F 006E5          PUSHAB  PATHDESC                        ;
0000V   CF              04  FB 006E9          CALLS   #4, PATHNAME_TO_PRIMARY         ;
  24    AE              50  D0 006EE          MOVL    R0, PRIMPTR                     ;
                  24    AE  DD 006F2 81$:     PUSHL   PRIMPTR                         ; 7806
                      01C9  31 006F5          BRW     120$                            ;
                        7E  D4 006F8 82$:     CLRL    -(SP)                           ; 7822
                        56  DD 006FA          PUSHL   LAST_OPERAND                    ;
                FF30    CD  9F 006FC          PUSHAB  PATHDESC                        ;
0000V   CF              03  FB 00700          CALLS   #3, APPEND_TO_PATHNAME          ;
                FE60    CD  9F 00705          PUSHAB  SAVED_PATHDESC                  ; 7823
                        57  DD 00709          PUSHL   PLIPTR                          ; 7824
                  48    AE  9F 0070B          PUSHAB  SUBSCR_DESC                     ; 7823
                FF30    CD  9F 0070E          PUSHAB  PATHDESC                        ;
0000V   CF              04  FB 00712          CALLS   #4, PATHNAME_TO_PRIMARY         ;
```

```
                57          50  D0 00717          MOVL    R0, PLIPTR
                06    07    A7  91 0071A          CMPB    7(PLIPTR), #6                   ; 7825
                            09  12 0071E          BNEQ    83$
                10    06    A7  91 00720          CMPB    6(PLIPTR), #16                  ; 7826
                            03  12 00724          BNEQ    83$
                          014C  31 00726          BRW     112$
                          28    AE 9F 00729 83$:  PUSHAB  NAME                           ; 7831
                        FF30    CD 9F 0072C       PUSHAB  PATHDESC
00000000G       00              02  FB 00730     CALLS   #2, DBG$NPATHDESC_TO_CS
                          28    AE DD 00737       PUSHL   NAME                           ; 7832
                        0129    31 0073A          BRW     111$
                          7E    D4 0073D 84$:     CLRL    -(SP)                          ; 7908
                          12    11 0073F          BRB     89$
                          08    A6 9F 00741 85$:  PUSHAB  8(LAST_OPERAND)                ; 7917
                          5A    DD 00744          PUSHL   R10
0000V   CF                      02  FB 00746     CALLS   #2, GET_RECORD_COMPONENT
                          11    11 0074B 86$:     BRB     90$
                          01    DD 0074D 87$:     PUSHL   #1                             ; 7926
                          02    11 0074F          BRB     89$
                          02    DD 00751 88$:     PUSHL   #2                             ; 7935
                          56    DD 00753 89$:     PUSHL   LAST_OPERAND
                        FF30    CD 9F 00755       PUSHAB  PATHDESC
0000V   CF                      03  FB 00759     CALLS   #3, APPEND_TO_PATHNAME
                        F908    31 0075E 90$:     BRW     4$                             ; 7064
                          08    A6 9F 00761 91$:  PUSHAB  8(LAST_OPERAND)                ; 7947
                          5A    DD 00764          PUSHL   R10
0000V   CF                      02  FB 00766     CALLS   #2, GET_RECORD_COMPONENT
                        0148    31 0076B          BRW     117$                           ; 7948
                          01    DD 0076E 92$:     PUSHL   #1                             ; 7958
                          02    11 00770          BRB     94$
                          7E    D4 00772 93$:     CLRL    -(SP)                          ; 7968
                          56    DD 00774 94$:     PUSHL   LAST_OPERAND
                        FF30    CD 9F 00776       PUSHAB  PATHDESC
0000V   CF                      03  FB 0077A     CALLS   #3, APPEND_TO_PATHNAME
                        009A    31 0077F          BRW     107$                           ; 7969
                          08    A6 9F 00782 95$:  PUSHAB  8(LAST_OPERAND)                ; 7982
                          5A    DD 00785          PUSHL   R10
0000V   CF                      02  FB 00787     CALLS   #2, GET_RECORD_COMPONENT
                        0130    31 0078C          BRW     119$                           ; 7983
                          01    DD 0078F 96$:     PUSHL   #1                             ; 7997
                          56    DD 00791          PUSHL   LAST_OPERAND
                        FF30    CD 9F 00793       PUSHAB  PATHDESC
0000V   CF                      03  FB 00797     CALLS   #3, APPEND_TO_PATHNAME
                        FE60    CD 9F 0079C       PUSHAB  SAVED_PATHDESC                 ; 7998
                          57    DD 007A0          PUSHL   PLIPTR                         ; 7999
                          48    AE 9F 007A2       PUSHAB  SUBSCR_DESC                    ; 7998
                        FF30    CD 9F 007A5       PUSHAB  PATHDESC
0000V   CF                      04  FB 007A9     CALLS   #4, PATHNAME_TO_PRIMARY
                57              50  D0 007AE     MOVL    R0, PLIPTR
                06    07    A7  91 007B1          CMPB    7(PLIPTR), #6                   ; 8000
                            09  12 007B5          BNEQ    97$
                10    06    A7  91 007B7          CMPB    6(PLIPTR), #16                  ; 8001
                            03  12 007BB          BNEQ    97$
                          00B5  31 007BD          BRW     112$
                          2C    AE 9F 007C0 97$:  PUSHAB  NAME                           ; 8006
                        FF30    CD 9F 007C3       PUSHAB  PATHDESC
00000000G       00              02  FB 007C7     CALLS   #2, DBG$NPATHDESC_TO_CS
```

```
                                    2C   AE   DD 007CE             PUSHL   NAME                                    ; 8007
                                  0092   31   007D1             BRW     111$                                    ;
                                    01   DD   007D4 98$:         PUSHL   #1                                      ; 8027
                                  FD86   31   007D6             BRW     59$                                     ;
                                    38   AE   DD 007D9 99$:     PUSHL   PATHSTRING                              ; 8050
                                    01   DD   007DC             PUSHL   #1                                      ;
                              000281F8   8F   DD 007DE             PUSHL   #164344                                 ;
                00000000G   00           03   FB 007E4 100$:     CALLS   #3, LIB$SIGNAL                          ;
                                    58   DD   007EB 101$:        PUSHL   TOKEN                                   ; 8053
                                    38   AE   DD 007ED            PUSHL   TYPEID                                  ;
                00000000G   00           02   FB 007F0            CALLS   #2, DBG$EVAL_ADA_TICK                   ;
                      24    AE           50   D0 007F7 102$:      MOVL    R0, PRIMPTR                             ;
                      18    BC           01   D0 007FB            MOVL    #1, @RET_OPERAND_FLAG                   ; 8061
                                    0A   11   007FF             BRB     104$                                    ; 8026
                                    08   A6   9F 00801 103$:     PUSHAB  8(LAST_OPERAND)                         ; 8074
                                    5A   DD   00804             PUSHL   R10                                     ;
                     0000V  CF           02   FB 00806            CALLS   #2, GET_RECORD_COMPONENT                ;
                                  00BB   31   0080B 104$:        BRW     122$                                    ; 8073
                                    01   DD   0080E 105$:        PUSHL   #1                                      ; 8084
                                  FE0B   31   00810             BRW     70$                                     ; 8096
                                    5A   DD   00813 106$:        PUSHL   R10                                     ; 8109
                     0000V  CF           01   FB 00815            CALLS   #1, FIX_UP_PRIMARY                      ;
                                    77   11   0081A             BRB     113$                                    ;
                                    40   AE   9F 0081C 107$:     PUSHAB  SUBSCR_DESC                             ; 8136
                                  FF30   CD   9F 0081F             PUSHAB  PATHDESC                                ;
                     0000V  CF           02   FB 00823            CALLS   #2, SAVE_SUBSCRIPTS                     ;
                                    69   11   00828             BRB     113$                                    ; 7064
                                    5A   DD   0082A 108$:        PUSHL   R10                                     ; 8149
                     0000V  CF           01   FB 0082C            CALLS   #1, FIX_UP_PRIMARY                      ;
                                  008B   31   00831             BRW     119$                                    ; 8150
                                  FE60   CD   9F 00834 109$:     PUSHAB  SAVED_PATHDESC                          ; 8164
                                    57   DD   00838             PUSHL   PLIPTR                                  ; 8165
                                    48   AE   9F 0083A             PUSHAB  SUBSCR_DESC                             ; 8164
                                  FF30   CD   9F 0083D             PUSHAB  PATHDESC                                ;
                     0000V  CF           04   FB 00841            CALLS   #4, PATHNAME_TO_PRIMARY                 ;
                            57           50   D0 00846            MOVL    R0, PLIPTR                              ;
                            06     07    A7   91 00849            CMPB    7(PLIPTR), #6                           ; 8166
                                    06   12   0084D             BNEQ    110$                                    ;
                            10     06    A7   91 0084F            CMPB    6(PLIPTR), #16                          ; 8167
                                    20   13   00853             BEQL    112$                                    ;
                                    3C   AE   9F 00855 110$:     PUSHAB  NAME                                    ; 8172
                                  FF30   CD   9F 00858             PUSHAB  PATHDESC                                ;
                00000000G   00           02   FB 0085C            CALLS   #2, DBG$NPATHDESC_TO_CS                 ;
                                    3C   AE   DD 00863             PUSHL   NAME                                    ; 8173
                                    01   DD   00866 111$:        PUSHL   #1                                      ;
                              00028CA0   8F   DD 00868             PUSHL   #167072                                 ;
                00000000G   00           03   FB 0086E            CALLS   #3, LIB$SIGNAL                          ;
                                    24   AE   D4 00875 112$:     CLRL    PRIMPTR                                 ; 8178
                                    08   AE   D4 00878             CLRL    NUMERIC_PATHNAME                        ; 8179
0DD0  8F      00            6E           00   2C 0087B            MOVC5   #0, (SP), #0, #208, PATHDESC            ; 8180
                                  FF30   CD   00882                                                              ;
                            5B    FF38   CD   9E 00885            MOVAB   PATHDESC+8, PATHVECTOR                  ; 8181
0270  8F      00            6E           00   2C 0088A            MOVC5   #0, (SP), #0, #624, SUBSCR_DESC         ; 8182
                                    40   AE   00891                                                              ;
                                    31   11   00893 113$:        BRB     121$                                    ; 7064
                                    5A   DD   00895 114$:        PUSHL   R10                                     ; 8194
                     0000V  CF           01   FB 00897            CALLS   #1, FIX_UP_PRIMARY                      ;
```

```
                                    28  11 0089C                BRB     122$                                    ; 8193
                                FE60    CD  9F 0089E 115$:      PUSHAB  SAVED_PATHDESC                          ; 8204
                                    57  DD 008A2              PUSHL   PLIPTR                                  ; 8205
                                  48    AE  9F 008A4          PUSHAB  SUBSCR_DESC                             ; 8204
                                FF30    CD  9F 008A7          PUSHAB  PATHDESC                                ;
                          0000V CF      04  FB 008AB          CALLS   #4, PATHNAME_TO_PRIMARY                 ;
                            24  AE      50  D0 008B0 116$:     MOVL    R0, PRIMPTR                             ;
                                    13  11 008B4              BRB     122$                                    ; 8203
                                    5A  DD 008B6 117$:      PUSHL   R10                                     ; 8224
                          0000V CF      01  FB 008B8 118$:      CALLS   #1, GET_SUBSCRIPTS                      ;
                                    07  11 008BD              BRB     121$                                    ;
                                    5A  DD 008BF 119$:      PUSHL   R10                                     ; 8233
                          0000V CF      01  FB 008C1 120$:      CALLS   #1, GET_DEREFERENCE                     ;
                                F7A0    51 008C6 121$:      BRW     4$                                      ;
                            14  BC      24  AE  D0 008C9 122$:      MOVL    PRIMPTR, @RET_TOKEN             ; 8259
                                        04  008CE              RET                                             ; 8262

; Routine Size:  2255 bytes,    Routine Base:  DBG$CODE + 198E
```

```
;  8155          8263  1 ROUTINE APPEND_TO_PATHNAME(PATHDESCR, TOKEN, COMPONENT_KIND): NOVALUE =
;  8156          8264  1 !
;  8157          8265  1 ! FUNCTION
;  8158          8266  1 !       This routine appends a pathname component to an existing Pathname
;  8159          8267  1 !       Descriptor.  It is called by the Primary Parser during the parsing of
;  8160          8268  1 !       the pathname part of Primary Symbols (e.g., A\B\C) to build up the
;  8161          8269  1 !       Pathname Descriptor which must eventually be passed to GETSYMBOL to
;  8162          8270  1 !       get the symbol's SYMID.
;  8163          8271  1 !
;  8164          8272  1 ! INPUTS
;  8165          8273  1 !       PATHDESCR - A pointer to a Pathname Descriptor to which a new pathname
;  8166          8274  1 !                   component should be appended.
;  8167          8275  1 !
;  8168          8276  1 !       TOKEN     - A pointer to the Lexical Token Entry for the identifier to
;  8169          8277  1 !                   be appended to the pathname descriptor.
;  8170          8278  1 !
;  8171          8279  1 !       COMPONENT_KIND - This tells us what kind of pathname component we
;  8172          8280  1 !                   are dealing with. It can have one of the following values:
;  8173          8281  1 !                   NOT_REC_COMP (0) - An ordinary pathname component to be
;  8174          8282  1 !                                      appended onto the end of the pathname,
;  8175          8283  1 !                                      e.g., "A" and "B" would be of this kind
;  8176          8284  1 !                                      in the pathname "A\B.C"
;  8177          8285  1 !                   REC_COMP (1)     - In the above example, "C" comes in as
;  8178          8286  1 !                                      a record component. It is appended to
;  8179          8287  1 !                                      the pathname and the TOTCNT is incremented
;  8180          8288  1 !                                      but not the PTHCNT field.
;  8181          8289  1 !                   COB_REC_COMP (2) - In COBOL, record components come in
;  8182          8290  1 !                                      first, e.g., "C of B of A", so we have
;  8183          8291  1 !                                      to treat them differently when we are
;  8184          8292  1 !                                      building the pathname.
;  8185          8293  1 !
;  8186          8294  1 ! OUTPUTS
;  8187          8295  1 !       The identifier specified by TOKEN is added to the end of the Pathname
;  8188          8296  1 !               Descriptor pointed to by PATHDESCR.
;  8189          8297  1 !
;  8190          8298  1
;  8191          8299  2     BEGIN
;  8192          8300  2
;  8193          8301  2     MAP
;  8194          8302  2         PATHDESCR: REF PTH$PATHNAME,     ! Pointer to Pathname Descriptor
;  8195          8303  2         TOKEN: REF TOKEN$ENTRY;          ! Pointer to Identifier Token Entry
;  8196          8304  2
;  8197          8305  2     LOCAL
;  8198          8306  2         PATHVECTOR: REF VECTOR[,LONG];   ! Pointer to pathname vector in the
;  8199          8307  2                                          !      PATHDESCR Pathname Descriptor
;  8200          8308  2
;  8201          8309  2
;  8202          8310  2
;  8203          8311  2     ! Make sure we have a valid Identifier Lexical Token Entry.  Also make sure
;  8204          8312  2     ! it will fit in the Pathname Descriptor.
;  8205          8313  2     !
;  8206          8314  2     IF .TOKEN[TOKEN$W_CODE] NEQ TOKEN$K_IDENTIFIER
;  8207          8315  2     THEN
;  8208          8316  2         SIGNAL(DBG$_ILLPATHELEM, 1, TOKEN[TOKEN$B_LENGTH]);
;  8209          8317  2
;  8210          8318  2     IF .PATHDESCR[PTH$B_PATHCNT] GEQ DBG$K_MAX_PATHNAME
;  8211          8319  2     THEN
```

```
;  8212          8320  2          SIGNAL(DBG$_PATHTOOLONG, 1, TOKEN[TOKEN$B_LENGTH]);
;  8213          8321  2
;  8214          8322  2      PATHVECTOR = PATHDESCR[PTH$A_PATHVECTOR];
;  8215          8323  2
;  8216          8324  2      ! Handle a non-COBOL record component.
;  8217          8325  2      !
;  8218          8326  2      IF .COMPONENT_KIND EQL REC_COMP
;  8219          8327  2      THEN
;  8220          8328  3          BEGIN
;  8221          8329  3
;  8222          8330  3          ! Append the new pathname component to the Pathname Descriptor
;  8223          8331  3          ! and then just return.
;  8224          8332  3          !
;  8225          8333  3          PATHVECTOR[.PATHDESCR[PTH$B_TOTCNT]] = TOKEN[TOKEN$B_LENGTH];
;  8226          8334  3          PATHDESCR[PTH$B_TOTCNT] = .PATHDESCR[PTH$B_TOTCNT] + 1;
;  8227          8335  3          RETURN;
;  8228          8336  2          END;
;  8229          8337  2
;  8230          8338  2      ! Handle the case where we are adding a record component name in COBOL.
;  8231          8339  2      ! Since these come in reverse order, previous ones must be pushed down.
;  8232          8340  2      !
;  8233          8341  2      IF .COMPONENT_KIND EQL COB_REC_COMP
;  8234          8342  2      THEN
;  8235          8343  3          BEGIN
;  8236          8344  3
;  8237          8345  3          ! Check that we have not seen any pathname components yet.
;  8238          8346  3          !
;  8239          8347  3          IF .PATHDESCR[PTH$B_PATHCNT] NEQ 0
;  8240          8348  3          THEN
;  8241          8349  3              $DBG_ERROR('DBGPARSER\APPEND_TO_PATHNAME 10');
;  8242          8350  3
;  8243          8351  3          ! Push the previous components.
;  8244          8352  3          !
;  8245          8353  3          DECR I FROM .PATHDESCR[PTH$B_TOTCNT] TO 1 DO
;  8246          8354  3              PATHVECTOR[.I] = .PATHVECTOR[.I-1];
;  8247          8355  3
;  8248          8356  3          ! Add the new component.
;  8249          8357  3          !
;  8250          8358  3          PATHVECTOR[0] = TOKEN[TOKEN$B_LENGTH];
;  8251          8359  3
;  8252          8360  3          ! Bump the count and we are done.
;  8253          8361  3          !
;  8254          8362  3          PATHDESCR[PTH$B_TOTCNT] = .PATHDESCR[PTH$B_TOTCNT] + 1;
;  8255          8363  3          RETURN;
;  8256          8364  2          END;
;  8257          8365  2
;  8258          8366  2      ! Finally handle the ordinary pathname components.
;  8259          8367  2      ! We want to insert the given component after previous pathname components
;  8260          8368  2      ! but before any record components we have already picked up.
;  8261          8369  2      !
;  8262          8370  2      IF .COMPONENT_KIND EQL NOT_REC_COMP
;  8263          8371  2      THEN
;  8264          8372  3          BEGIN
;  8265          8373  3
;  8266          8374  3          ! We want to insert the name at the PATHCNT position.
;  8267          8375  3          ! Push down lower names (record component names).
;  8268          8376  3          !
```

```
;  8269          8377  3          DECR I FROM .PATHDESCR[PTH$B_TOTCNT] TO .PATHDESCR[PTH$B_PATHCNT]+1 DO
;  8270          8378  3              PATHVECTOR[.I] = .PATHVECTOR[.I-1];
;  8271          8379  3
;  8272          8380  3          ! Insert the given name, increment the counts, and return.
;  8273          8381  3          !
;  8274          8382  3          PATHVECTOR[.PATHDESCR[PTH$B_PATHCNT]] = TOKEN[TOKEN$B_LENGTH];
;  8275          8383  3          PATHDESCR[PTH$B_PATHCNT] = .PATHDESCR[PTH$B_PATHCNT] + 1;
;  8276          8384  3          PATHDESCR[PTH$B_TOTCNT] = .PATHDESCR[PTH$B_TOTCNT] + 1;
;  8277          8385  3          RETURN;
;  8278          8386  2          END;
;  8279          8387  1      END;


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

45 50 50 41 5C 52 45 53 52 41 50 47 42 44 1F  031DB P.AXL:  .ASCII  <31>\DBGPARSER\<92>\APPEND_TO_PATHNAME\      ;
   45 4D 41 4E 48 54 41 50 5F 4F 54 5F 44 4E  031EA                                                            ;
                                 30 31 20  031F8           .ASCII  \ 10\                                       ;


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

                                  007C 00000 APPEND_TO_PATHNAME:
                                                        .WORD   Save R2,R3,R4,R5,R6                             ; 8263
                          56 00000000G   00  9E 00002   MOVAB   LIB$SIGNAL, R6                                  ;
                          54         08  AC  D0 00009   MOVL    TOKEN, R4                                       ; 8314
                          01         02  A4  B1 0000D   CMPW    2(R4), #1                                       ;
                                         0E  13 00011   BEQL    1$                                              ;
                                     08  A4  9F 00013   PUSHAB  8(R4)                                           ; 8316
                                         01  DD 00016   PUSHL   #1                                              ;
                             0002898A    8F  DD 00018   PUSHL   #166282                                          ;
                          66             03  FB 0001E   CALLS   #3, LIB$SIGNAL                                  ;
                          53         04  AC  D0 00021 1$:  MOVL    PATHDESCR, R3                                   ; 8318
                          32         01  A3  91 00025   CMPB    1(R3), #50                                      ;
                                         0E  1F 00029   BLSSU   2$                                              ;
                                     08  A4  9F 0002B   PUSHAB  8(R4)                                           ; 8320
                                         01  DD 0002E   PUSHL   #1                                              ;
                             000289D2    8F  DD 00030   PUSHL   #166354                                          ;
                          66             03  FB 00036   CALLS   #3, LIB$SIGNAL                                  ;
                          52         08  A3  9E 00039 2$:  MOVAB   8(R3), PATHVECTOR                              ; 8322
                          01         0C  AC  D1 0003D   CMPL    COMPONENT_KIND, #1                              ; 8326
                                         0A  12 00041   BNEQ    3$                                              ;
                          50             63  9A 00043   MOVZBL  (R3), R0                                       ; 8333
                        6240         08  A4  9E 00046   MOVAB   8(R4), (PATHVECTOR)[R0]                         ;
                                         59  11 0004B   BRB     10$                                             ; 8334
                          02         0C  AC  D1 0004D 3$:  CMPL    COMPONENT_KIND, #2                              ; 8341
                                         2C  12 00051   BNEQ    7$                                              ;
                                     01  A3  95 00053   TSTB    1(R3)                                           ; 8347
                                         11  13 00056   BEQL    4$                                              ;
                             00000000'   EF  9F 00058   PUSHAB  P.AXL                                           ; 8349
                                         01  DD 0005E   PUSHL   #1                                              ;
                             00028362    8F  DD 00060   PUSHL   #164706                                          ;
                          66             03  FB 00066   CALLS   #3, LIB$SIGNAL                                  ;
                          50             63  9A 00069 4$:  MOVZBL  (R3), I                                      ; 8354
                                         50  D6 0006C   INCL    I                                               ;
```

```
                          06  11 0006E            BRB     6$                                                      ;
                6240    FC A240  D0 00070 5$:      MOVL    -4(PATHVECTOR)[I], (PATHVECTOR)[I]                      ;
                  F7        50  F5 00076 6$:      SOBGTR  I, 5$                                                   ;
                  62    08  A4  9E 00079          MOVAB   8(R4), (PATHVECTOR)                                     ; 8358
                            27  11 0007D          BRB     10$                                                     ; 8362
                        0C  AC  D5 0007F 7$:      TSTL    COMPONENT_KIND                                          ; 8370
                            24  12 00082          BNEQ    11$                                                     ;
                  51    01  A3  9A 00084          MOVZBL  1(R3), R1                                               ; 8377
                  55    01  A1  9E 00088          MOVAB   1(R1), R5                                               ;
                  50        63  9A 0008C          MOVZBL  (R3), I                                                 ; 8378
                            08  11 0008F          BRB     9$                                                      ;
                6240    FC A240  D0 00091 8$:      MOVL    -4(PATHVECTOR)[I], (PATHVECTOR)[I]                      ;
                            50  D7 00097          DECL    I                                                       ;
                  55        50  D1 00099 9$:      CMPL    I, R5                                                   ;
                            F3  18 0009C          BGEQ    8$                                                      ;
                6241    08  A4  9E 0009E          MOVAB   8(R4), (PATHVECTOR)[R1]                                 ; 8382
                    01  A3  96 000A3            INCB    1(R3)                                                   ; 8383
                        63  96 000A6 10$:        INCB    (R3)                                                    ; 8384
                            04 000A8 11$:        RET                                                             ; 8387

; Routine Size:  169 bytes,    Routine Base:  DBG$CODE + 225D
```

```
;  8281          8388  1 ROUTINE CHECK_UPSCOPE(SYMID, TYPEID, PRIMPTR, DEPTH) =
;  8282          8389  1 !
;  8283          8390  1 ! FUNCTION
;  8284          8391  1 !       This routine is used to implement "incomplete data qualification"
;  8285          8392  1 !       in language BASIC. For example, a user can abbreviate A::B::C with
;  8286          8393  1 !       A::C. We are passed in a SYMID for the record component ("C" in
;  8287          8394  1 !       the above example), and a TYPEID for the record ("A" in the above
;  8288          8395  1 !       example. This routine determines whether we can get to the record
;  8289          8396  1 !       by going upscope from the component. If so, it returns TRUE and
;  8290          8397  1 !       modifies the Primary to include the intervening component
;  8291          8398  1 !       selection.
;  8292          8399  1 !
;  8293          8400  1 ! INPUTS
;  8294          8401  1 !       SYMID   - SYMID for the record component.
;  8295          8402  1 !       TYPEID  - TYPEID for the record.
;  8296          8403  1 !       PRIMPTR - Pointer to the Primary being constructed.
;  8297          8404  1 !       DEPTH   - recursion depth
;  8298          8405  1 !
;  8299          8406  1 ! OUTPUTS
;  8300          8407  1 !       The value TRUE is returned if the component selection is valid.
;  8301          8408  1 !       In this case, the Primary is modified to include the intervening
;  8302          8409  1 !       component selection.
;  8303          8410  1 !       The value FALSE is returned if the component is not a valid
;  8304          8411  1 !       component for this record.
;  8305          8412  1 !
;  8306          8413  2     BEGIN
;  8307          8414  2     MAP
;  8308          8415  2         SYMID: REF RST$ENTRY,
;  8309          8416  2         TYPEID: REF RST$ENTRY,
;  8310          8417  2         PRIMPTR: REF DBG$PRIMARY;
;  8311          8418  2
;  8312          8419  2     LOCAL
;  8313          8420  2         I,
;  8314          8421  2         COMPSYMID,
;  8315          8422  2         FCODE,
;  8316          8423  2         NEW_TYPEID,
;  8317          8424  2         NODEPTR: REF DBG$PRIM_NODE,
;  8318          8425  2         TYPCOMPLST: REF VECTOR[,LONG],
;  8319          8426  2         TYPREFTBL: REF VECTOR[,LONG],
;  8320          8427  2         U_SYMID: REF RST$ENTRY;
;  8321          8428  2
;  8322          8429  2     DBG$GL_CURRENT_PRIMARY = .PRIMPTR;
;  8323          8430  2
;  8324          8431  2     ! Check that the RST entry upscope from the given
;  8325          8432  2     ! record component SYMID is a Type RST Entry.
;  8326          8433  2     ! Then check whether it matches the given TYPEID for the record.
;  8327          8434  2     !
;  8328          8435  2     U_SYMID = .SYMID[RST$L_UPSCOPEPTR];
;  8329          8436  2     IF .U_SYMID[RST$B_KIND] NEQ RST$K_TYPE
;  8330          8437  2     THEN
;  8331          8438  2         RETURN FALSE;
;  8332          8439  2     IF .U_SYMID EQL .TYPEID
;  8333          8440  2     THEN
;  8334          8441  3         BEGIN
;  8335          8442  3
;  8336          8443  3         ! The TYPEID matches, so the record component SYMID is valid.
;  8337          8444  3         ! In this case, we do want to perform the component selection.
```

```
;  8338     8445  3          ! Modify the Primary to include the record component selection.
;  8339     8446  3          !
;  8340     8447  3          NODEPTR = .PRIMPTR[DBG$L_PRIM_BLINK];
;  8341     8448  3          NODEPTR[DBG$V_PNODE_EVAL] = TRUE;
;  8342     8449  3          TYPCOMPLST = TYPEID[RST$A_TYPCOMPLST];
;  8343     8450  3
;  8344     8451  3          ! Determine the "component index". This "PNREC_INDEX" field gets
;  8345     8452  3          ! used in determining the logical successor.
;  8346     8453  3          !
;  8347     8454  3          INCR I FROM 0 TO .TYPEID[RST$L_TYPCOMPCNT] - 1 DO
;  8348     8455  4              BEGIN
;  8349     8456  4              COMPSYMID = .TYPCOMPLST[.I];
;  8350     8457  4
;  8351     8458  4
;  8352     8459  4              ! If this component is the one we seek, set its index into the Record
;  8353     8460  4              ! Sub-Node and leave the loop.
;  8354     8461  4              !
;  8355     8462  4              IF .SYMID EQL .COMPSYMID
;  8356     8463  4              THEN
;  8357     8464  5                  BEGIN
;  8358     8465  5                  NODEPTR[DBG$W_PNREC_INDEX] = .I + 1;
;  8359     8466  5                  EXITLOOP;
;  8360     8467  4                  END;
;  8361     8468  3              END;
;  8362     8469  3
;  8363     8470  3          ! Check for FCODE of record - otherwise, it is not valid to skip
;  8364     8471  3          ! this component.
;  8365     8472  3          !
;  8366     8473  3          DBG$STA_SYMTYPE(.SYMID, FCODE, NEW_TYPEID);
;  8367     8474  3          IF .FCODE NEQ RST$K_TYPE_RECORD
;  8368     8475  3          THEN
;  8369     8476  3              RETURN FALSE;
;  8370     8477  3
;  8371     8478  3          ! Finally append another Primary Descriptor Sub-Node for the selected
;  8372     8479  3          ! record component.  Then return.
;  8373     8480  3          !
;  8374     8481  3          DBG$BUILD_PRIMARY_SUBNODE(.PRIMPTR, RST$K_TYPCOMP, .SYMID,
;  8375     8482  3                                    .FCODE, .NEW_TYPEID, 0);
;  8376     8483  3          RETURN TRUE;
;  8377     8484  2          END;
;  8378     8485  2
;  8379     8486  2      ! The immediate upscope pointer did not match. In this case, go further
;  8380     8487  2      ! upscope. We do this by getting the Type Reference table for this
;  8381     8488  2      ! Type RST entry. For each SYMID in the table, we recursively call
;  8382     8489  2      ! the CHECK_UPSCOPE routine.
;  8383     8490  2      !
;  8384     8491  2      TYPREFTBL = .U_SYMID[RST$L_TYPREFTBL];
;  8385     8492  2      I = 0;
;  8386     8493  2      WHILE .TYPREFTBL[.I] NEQ 0 DO
;  8387     8494  3          BEGIN
;  8388     8495  3          U_SYMID = .TYPREFTBL[.I];
;  8389     8496  3          IF CHECK_UPSCOPE(.U_SYMID, .TYPEID, .PRIMPTR, .DEPTH+1)
;  8390     8497  3          THEN
;  8391     8498  4              BEGIN
;  8392     8499  4
;  8393     8500  4              ! If we are at the top level of recursion, then do not modify
;  8394     8501  4              ! the Primary here. This is because it will get done in
```

```
;  8395          8502  4          ! GET_RECORD_COMPONENT.
;  8396          8503  4          !
;  8397          8504  4          IF .DEPTH EQL 0
;  8398          8505  4          THEN
;  8399          8506  4              RETURN TRUE;
;  8400          8507  4
;  8401          8508  4          ! The TYPEID matches, so the record component SYMID is valid.
;  8402          8509  4          ! In this case, we do want to perform the component selection.
;  8403          8510  4          ! Modify the Primary to include the record component selection.
;  8404          8511  4          !
;  8405          8512  4          NODEPTR = .PRIMPTR[DBG$L_PRIM_BLINK];
;  8406          8513  4          NODEPTR[DBG$V_PNODE_EVAL] = TRUE;
;  8407          8514  4          TYPCOMPLST = TYPEID[RST$A_TYPCOMPLST];
;  8408          8515  4
;  8409          8516  4          ! Determine the "component index". This "PNREC_INDEX" field gets
;  8410          8517  4          ! used in determining the logical successor.
;  8411          8518  4          !
;  8412          8519  4          INCR I FROM 0 TO .TYPEID[RST$L_TYPCOMPCNT] - 1 DO
;  8413          8520  5              BEGIN
;  8414          8521  5              COMPSYMID = .TYPCOMPLST[.I];
;  8415          8522  5
;  8416          8523  5
;  8417          8524  5              ! If this component is the one we seek, set its index into the Record
;  8418          8525  5              ! Sub-Node and leave the loop.
;  8419          8526  5              !
;  8420          8527  5              IF .SYMID EQL .COMPSYMID
;  8421          8528  5              THEN
;  8422          8529  6                  BEGIN
;  8423          8530  6                  NODEPTR[DBG$W_PNREC_INDEX] = .I + 1;
;  8424          8531  6                  EXITLOOP;
;  8425          8532  5                  END;
;  8426          8533  4              END;
;  8427          8534  4
;  8428          8535  4              ! Check for FCODE of record - otherwise, it is not valid to skip
;  8429          8536  4              ! this component.
;  8430          8537  4              !
;  8431          8538  4              DBG$STA_SYMTYPE(.SYMID, FCODE, NEW_TYPEID);
;  8432          8539  4              IF .FCODE NEQ RST$K_TYPE_RECORD
;  8433          8540  4              THEN
;  8434          8541  4                  RETURN FALSE;
;  8435          8542  4
;  8436          8543  4
;  8437          8544  4              ! Finally append another Primary Descriptor Sub-Node for the selected
;  8438          8545  4              ! record component.  Then return.
;  8439          8546  4              !
;  8440          8547  4              DBG$BUILD_PRIMARY_SUBNODE(.PRIMPTR, RST$K_TYPCOMP, .SYMID,
;  8441          8548  4                                        .FCODE, .NEW_TYPEID, 0);
;  8442          8549  4              RETURN TRUE;
;  8443          8550  3              END;
;  8444          8551  3
;  8445          8552  3          I = .I + 1;
;  8446          8553  2          END;
;  8447          8554  2
;  8448          8555  2      ! We failed to find a path to the desired TYPEID. Return FALSE.
;  8449          8556  2      !
;  8450          8557  2      RETURN FALSE;
;  8451          8558  1      END;
```

```
                                 0FFC 00000 CHECK_UPSCOPE:
                                                .WORD   Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11                    ; 8388
                          5E        08  C2 00002         SUBL2   #8, SP                                          ;
                          56    0C  AC  D0 00005         MOVL    PRIMPTR, R6                                     ; 8429
                00000000G 00        56  D0 00009         MOVL    R6, DBG$GL_CURRENT_PRIMARY                      ;
                          58    04  AC  D0 00010         MOVL    SYMID, R8                                       ; 8435
                          57    10  A8  D0 00014         MOVL    16(R8), U_SYMID                                 ;
                          07    14  A7  91 00018         CMPB    20(U_SYMID), #7                                 ; 8436
                                    03  13 0001C         BEQL    1$                                              ;
                                00A6  31 0001E         BRW     12$                                             ;
                          55    08  AC  D0 00021 1$:     MOVL    TYPEID, R5                                      ; 8439
                          55        57  D1 00025         CMPL    U_SYMID, R5                                     ;
                                    21  12 00028         BNEQ    4$                                              ;
                          53    18  A6  D0 0002A         MOVL    24(R6), NODEPTR                                 ; 8447
                       0A A3        01  88 0002E         BISB2   #1, 10(NODEPTR)                                 ; 8448
                          52    2C  A5  9E 00032         MOVAB   44(R5), TYPCOMPLST                              ; 8449
                          50        01  CE 00036         MNEGL   #1, I                                           ; 8456
                                    09  11 00039         BRB     3$                                              ;
                          59      6240  D0 0003B 2$:     MOVL    (TYPCOMPLST)[I], COMPSYMID                      ;
                          59        58  D1 0003F         CMPL    R8, COMPSYMID                                   ; 8462
                                    48  13 00042         BEQL    7$                                              ;
                    F2    50    28  A5  F2 00044 3$:     AOBLSS  40(R5), I, 2$                                   ; 8454
                                    4D  11 00049         BRB     9$                                              ; 8473
                          5A    1C  A7  D0 0004B 4$:     MOVL    28(U_SYMID), TYPREFTBL                          ; 8491
                                    54  D4 0004F         CLRL    I                                               ; 8492
                    5B        10  AC      01  C1 00051     ADDL3   #1, DEPTH, R11                                  ; 8496
                                  6A44  D5 00056 5$:     TSTL    (TYPREFTBL)[I]                                  ; 8493
                                    6C  13 00059         BEQL    12$                                             ;
                          57        6A44  D0 0005B       MOVL    (TYPREFTBL)[I], U_SYMID                         ; 8495
                                    5B  DD 0005F         PUSHL   R11                                             ; 8496
                          7E        55  7D 00061         MOVQ    R5, -(SP)                                       ;
                                    57  DD 00064         PUSHL   U_SYMID                                         ;
                          96    AF      04  FB 00066     CALLS   #4, CHECK_UPSCOPE                               ;
                          56        50  E9 0006A         BLBC    R0, 11$                                         ;
                                10  AC  D5 0006D         TSTL    DEPTH                                           ; 8504
                                    4D  13 00070         BEQL    10$                                             ;
                          53    18  A6  D0 00072         MOVL    24(R6), NODEPTR                                 ; 8512
                       0A A3        01  88 00076         BISB2   #1, 10(NODEPTR)                                 ; 8513
                          52    2C  A5  9E 0007A         MOVAB   44(R5), TYPCOMPLST                              ; 8514
                          50        01  CE 0007E         MNEGL   #1, I                                           ; 8521
                                    10  11 00081         BRB     8$                                              ;
                          59      6240  D0 00083 6$:     MOVL    (TYPCOMPLST)[I], COMPSYMID                      ;
                          59        58  D1 00087         CMPL    R8, COMPSYMID                                   ; 8527
                                    07  12 0008A         BNEQ    8$                                              ;
                18 A3             50      01  A1 0008C 7$:     ADDW3   #1, I, 24(NODEPTR)                      ; 8530
                                    05  11 00091         BRB     9$                                              ; 8529
                    EB            50  28  A5  F2 00093 8$:     AOBLSS  40(R5), I, 6$                           ; 8519
                                    5E  DD 00098 9$:     PUSHL   SP                                              ; 8538
                                10  AE  9F 0009A         PUSHAB  FCODE                                           ;
                                    58  DD 0009D         PUSHL   R8                                              ;
                      00000000G 00        03  FB 0009F     CALLS   #3, DBG$STA_SYMTYPE                             ;
                          07    04  AE  D1 000A6         CMPL    FCODE, #7                                       ; 8539
```

```
                                 1B  12 000AA          BNEQ    12$                                             ;
                                 7E  D4 000AC          CLRL    -(SP)                                           ; 8547
                              04 AE  DD 000AE          PUSHL   NEW_TYPEID                                      ; 8548
                              0C AE  DD 000B1          PUSHL   FCODE                                           ;
                                 58  DD 000B4          PUSHL   R8                                              ; 8547
                                 0A  DD 000B6          PUSHL   #10                                             ;
                                 56  DD 000B8          PUSHL   R6                                              ;
                        DCA7     CF  FB 000BA          CALLS   #6, DBG$BUILD_PRIMARY_SUBNODE                   ;
                                 50  01 D0 000BF 10$:   MOVL    #1, R0                                          ; 8549
                                     04 000C2          RET                                                     ;
                                 54  D6 000C3 11$:     INCL    I                                               ; 8552
                                 8F  11 000C5          BRB     5$                                              ; 8493
                                 50  D4 000C7 12$:     CLRL    R0                                              ; 8558
                                     04 000C9          RET                                                     ;

; Routine Size:  202 bytes,    Routine Base:  DBG$CODE + 2306
```

```
;  8453          8559  1 ROUTINE CONSTANT_TO_VALDESCR(TOKEN) =
;  8454          8560  1 !
;  8455          8561  1 ! FUNCTION
;  8456          8562  1 !       This routine converts a Constant Lexical Token Entry into the corre-
;  8457          8563  1 !       sponding Value Descriptor.  For string and character constants, this
;  8458          8564  1 !       results in an ordinary Value Descriptor, but for numeric constants
;  8459          8565  1 !       it results in an "unconverted" Value Descriptor, i.e., a descriptor
;  8460          8566  1 !       in which the numeric constant is represented as the original input
;  8461          8567  1 !       character string.  The actual conversion of a numeric constant to its
;  8462          8568  1 !       binary representation is thus delayed until the appropriate precision
;  8463          8569  1 !       of the binary representation can be determined from context.
;  8464          8570  1 !
;  8465          8571  1 ! INPUTS
;  8466          8572  1 !       TOKEN   - A pointer to a Constant Lexical Token Entry.  This entry
;  8467          8573  1 !                 represents a numeric, string, or character constant of
;  8468          8574  1 !                 some sort.
;  8469          8575  1 !
;  8470          8576  1 ! OUTPUTS
;  8471          8577  1 !       A Value Descriptor is constructed for the constant and a pointer to
;  8472          8578  1 !                 that descriptor is returned as the routine value.
;  8473          8579  1 !
;  8474          8580  1
;  8475          8581  2     BEGIN
;  8476          8582  2
;  8477          8583  2     MAP
;  8478          8584  2         TOKEN: REF TOKEN$ENTRY;          ! Pointer to Lexical Token Entry
;  8479          8585  2
;  8480          8586  2     LOCAL
;  8481          8587  2         VALPTR: REF DBG$VALDESC;         ! Pointer to Value Descriptor we build
;  8482          8588  2
;  8483          8589  2
;  8484          8590  2
;  8485          8591  2     ! Build a skeleton Value Descriptor for the constant and copy the constant
;  8486          8592  2     ! character representation into that descriptor.
;  8487          8593  2     !
;  8488          8594  2     VALPTR = DBG$MAKE_SKELETON_DESC(DBG$K_VALUE_DESC, .TOKEN[TOKEN$B_LENGTH]);
;  8489          8595  2     VALPTR[DBG$B_DHDR_LANG] = .DBG$GB_LANGUAGE;
;  8490          8596  2     VALPTR[DBG$B_DHDR_KIND] = RST$K_DATA;
;  8491          8597  2     VALPTR[DBG$B_DHDR_FCODE] = RST$K_TYPE_ATOMIC;
;  8492          8598  2     VALPTR[DBG$V_DHDR_UNCVT] = TRUE;
;  8493          8599  2     VALPTR[DBG$B_VALUE_CLASS] = DSC$K_CLASS_S;
;  8494          8600  2     VALPTR[DBG$W_VALUE_LENGTH] = .TOKEN[TOKEN$B_LENGTH];
;  8495          8601  2     VALPTR[DBG$L_VALUE_POINTER] = VALPTR[DBG$A_VALUE_ADDRESS];
;  8496          8602  2     VALPTR[DBG$W_VALUE_TOKENCODE] = .TOKEN[TOKEN$W_CODE];
;  8497          8603  2     CH$MOVE(.TOKEN[TOKEN$B_LENGTH], TOKEN[TOKEN$A_NAME],
;  8498          8604  2                                     VALPTR[DBG$A_VALUE_ADDRESS]);
;  8499          8605  2
;  8500          8606  2
;  8501          8607  2     ! Determine what kind of constant this is and set the data type and the
;  8502          8608  2     ! "unconverted" bit accordingly.
;  8503          8609  2     !
;  8504          8610  2     CASE .TOKEN[TOKEN$W_CODE] FROM TOKEN$K_MIN_OPERAND TO TOKEN$K_MAX_OPERAND OF
;  8505          8611  2         SET
;  8506          8612  2
;  8507          8613  2
;  8508          8614  2         ! Handle character string constants.  Mark the data as type T.
;  8509          8615  2         !
```

```
;  8510     8616  2          [TOKEN$K_STRING]:
;  8511     8617  2              VALPTR[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_T;
;  8512     8618  2
;  8513     8619  2
;  8514     8620  2          ! Handle bit string constants.  Mark the data as type TF.
;  8515     8621  2          !
;  8516     8622  2          [TOKEN$K_BIT_STRING]:
;  8517     8623  2              VALPTR[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_V;
;  8518     8624  2
;  8519     8625  2
;  8520     8626  2          ! Handle decimal integer constants.  Mark the data as type L.
;  8521     8627  2          !
;  8522     8628  2          [TOKEN$K_INTEGER]:
;  8523     8629  2              VALPTR[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_L;
;  8524     8630  2
;  8525     8631  2
;  8526     8632  2          ! Handle hexadecimal integer constants.  Mark the data as type L.
;  8527     8633  2          !
;  8528     8634  2          [TOKEN$K_HEX_INTEGER]:
;  8529     8635  2              VALPTR[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_L;
;  8530     8636  2
;  8531     8637  2
;  8532     8638  2          ! Handle octal integer constants.  Mark the data as type L.
;  8533     8639  2          !
;  8534     8640  2          [TOKEN$K_OCT_INTEGER]:
;  8535     8641  2              VALPTR[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_L;
;  8536     8642  2
;  8537     8643  2
;  8538     8644  2          ! Handle binary integer constants.  Mark the data as type L.
;  8539     8645  2          !
;  8540     8646  2          [TOKEN$K_BIN_INTEGER]:
;  8541     8647  2              VALPTR[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_L;
;  8542     8648  2
;  8543     8649  2
;  8544     8650  2          ! Handle floating-point constants without exponents.  Mark the data
;  8545     8651  2          ! type as F.
;  8546     8652  2          !
;  8547     8653  2          [TOKEN$K_FLOATING]:
;  8548     8654  2              VALPTR[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_F;
;  8549     8655  2
;  8550     8656  2
;  8551     8657  2          ! Handle floating-point constants with E exponents.  Mark the data
;  8552     8658  2          ! type as F.
;  8553     8659  2          !
;  8554     8660  2          [TOKEN$K_EXP_E_FLOAT]:
;  8555     8661  2              VALPTR[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_F;
;  8556     8662  2
;  8557     8663  2
;  8558     8664  2          ! Handle floating-point constants with D exponents.  Mark the data
;  8559     8665  2          ! type as D.
;  8560     8666  2          !
;  8561     8667  2          [TOKEN$K_EXP_D_FLOAT]:
;  8562     8668  2              IF .DBG$GB_MOD_PTR[MODE_G_FLOATS]
;  8563     8669  2              THEN
;  8564     8670  3                  BEGIN
;  8565     8671  3                  VALPTR[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_G;
;  8566     8672  3                  VALPTR[DBG$W_VALUE_TOKENCODE] = TOKEN$K_EXP_G_FLOAT;
```

```
;  8567          8673  3                END
;  8568          8674  3
;  8569          8675  2            ELSE
;  8570          8676  2                VALPTR[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_D;
;  8571          8677  2
;  8572          8678  2
;  8573          8679  2        ! Handle floating-point constants with G exponents.  Mark the data
;  8574          8680  2        ! type as G.
;  8575          8681  2        !
;  8576          8682  2        [TOKEN$K_EXP_G_FLOAT]:
;  8577          8683  2            VALPTR[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_G;
;  8578          8684  2
;  8579          8685  2
;  8580          8686  2        ! Handle floating-point constants with Q exponents.  Mark the data
;  8581          8687  2        ! type as H.
;  8582          8688  2        !
;  8583          8689  2        [TOKEN$K_EXP_Q_FLOAT]:
;  8584          8690  2            VALPTR[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_H;
;  8585          8691  2
;  8586          8692  2
;  8587          8693  2        ! Handle pack decimal constants.  Mark the data type as P.
;  8588          8694  2        !
;  8589          8695  2        [TOKEN$K_PACK_DECIMAL]:
;  8590          8696  2            VALPTR[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_P;
;  8591          8697  2
;  8592          8698  2
;  8593          8699  2        ! Any other case is an internal error.
;  8594          8700  2        !
;  8595          8701  2        [INRANGE, OUTRANGE]:
;  8596          8702  2            $DBG_ERROR('DBGPARSER\CONSTANT_TO_VALDESC');
;  8597          8703  2
;  8598          8704  2        TES;
;  8599          8705  2
;  8600          8706  2
;  8601          8707  2    ! The Value Descriptor is constructed.  Return its address to the caller.
;  8602          8708  2    !
;  8603          8709  2    RETURN .VALPTR;
;  8604          8710  2
;  8605          8711  1    END;


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

53  4E  4F  43  5C  52  45  53  52  41  50  47  42  44  1D  031FB P.AXM:  .ASCII  <29>\DBGPARSER\<92>\CONSTANT_TO_VALDES\     ;
    53  45  44  4C  41  56  5F  4F  54  5F  54  4E  41  54  0320A                                                              ;
                                                        43  03218        .ASCII  \C\                                          ;


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

                                                    01FC 00000 CONSTANT_TO_VALDESCR:
                                                                .WORD   Save R2,R3,R4,R5,R6,R7,R8                           ; 8559
                                    58      04  AC  D0 00002         MOVL    TOKEN, R8                                       ; 8594
                                    7E      08  A8  9A 00006         MOVZBL  8(R8), -(SP)                                    ;
                                    7E      7A  8F  9A 0000A         MOVZBL  #122, -(SP)                                     ;
```

```
                                00000000G  00              02  FB 0000E          CALLS   #2, DBG$MAKE_SKELETON_DESC                ;
                                           56              50  D0 00015          MOVL    R0, VALPTR                                ;
                                     03    A6  00000000G   00  90 00018          MOVB    DBG$GB_LANGUAGE, 3(VALPTR)                ; 8595
                                     06    A6      0602    8F  B0 00020          MOVW    #1538, 6(VALPTR)                          ; 8597
                                     04    A6              20  88 00026          BISB2   #32, 4(VALPTR)                            ; 8598
                                           57        14    A6  9E 0002A          MOVAB   20(VALPTR), R7                            ; 8599
                                     03    A7              01  90 0002E          MOVB    #1, 3(R7)                                 ;
                                           67        08    A8  9B 00032          MOVZBW  8(R8), (R7)                               ; 8600
                                     18    A6        20    A6  9E 00036          MOVAB   32(VALPTR), 24(VALPTR)                    ; 8601
                                     10    A6        02    A8  B0 0003B          MOVW    2(R8), 16(VALPTR)                         ; 8602
                                           50        08    A8  9A 00040          MOVZBL  8(R8), R0                                 ; 8603
                              20   A6              09    A8      50  28 00044    MOVC3   R0, 9(R8), 32(VALPTR)                     ; 8604
                                    0F                    01      02  AF 0004A    CASEW   2(R8), #1, #15                            ; 8610
                         0043             0020             0037        0020      0004F 1$: .WORD   2$-1$,-                         ;
                         004F             0049             0049        0043      00057           3$-1$,-                           ;
                         0020             0043             0043        0070      0005F           2$-1$,-                           ;
                         0020             006A             0076        003D      00067           5$-1$,-                           ;
                                                                                                 5$-1$,-                           ;
                                                                                                 6$-1$,-                           ;
                                                                                                 6$-1$,-                           ;
                                                                                                 7$-1$,-                           ;
                                                                                                 10$-1$,-                          ;
                                                                                                 5$-1$,-                           ;
                                                                                                 5$-1$,-                           ;
                                                                                                 2$-1$,-                           ;
                                                                                                 4$-1$,-                           ;
                                                                                                 11$-1$,-                          ;
                                                                                                 9$-1$,-                           ;
                                                                                                 2$-1$                             ;
                                                00000000'  EF  9F 0006F 2$:     PUSHAB  P.AXM                                     ; 8702
                                                           01  DD 00075         PUSHL   #1                                        ;
                                                00028362   8F  DD 00077         PUSHL   #164706                                   ;
                                00000000G  00              03  FB 0007D         CALLS   #3, LIB$SIGNAL                            ;
                                                           43  11 00084         BRB     12$                                       ;
                                     02    A7              0E  90 00086 3$:     MOVB    #14, 2(R7)                                ; 8617
                                                           3D  11 0008A         BRB     12$                                       ;
                                     02    A7              01  90 0008C 4$:     MOVB    #1, 2(R7)                                 ; 8623
                                                           37  11 00090         BRB     12$                                       ;
                                     02    A7              08  90 00092 5$:     MOVB    #8, 2(R7)                                 ; 8647
                                                           31  11 00096         BRB     12$                                       ;
                                     02    A7              0A  90 00098 6$:     MOVB    #10, 2(R7)                                ; 8661
                                                           2B  11 0009C         BRB     12$                                       ;
                                           50  00000000G   00  D0 0009E 7$:     MOVL    DBG$GB_MOD_PTR, R0                        ; 8668
                                          0A         09    A0  E9 000A5         BLBC    9(R0), 8$                                 ;
                                     02    A7              1B  90 000A9         MOVB    #27, 2(R7)                                ; 8671
                                     10    A6              0F  B0 000AD         MOVW    #15, 16(VALPTR)                           ; 8672
                                                           16  11 000B1         BRB     12$                                       ; 8668
                                     02    A7              0B  90 000B3 8$:     MOVB    #11, 2(R7)                                ; 8676
                                                           10  11 000B7         BRB     12$                                       ; 8668
                                     02    A7              1B  90 000B9 9$:     MOVB    #27, 2(R7)                                ; 8683
                                                           0A  11 000BD         BRB     12$                                       ;
                                     02    A7              1C  90 000BF 10$:    MOVB    #28, 2(R7)                                ; 8690
                                                           04  11 000C3         BRB     12$                                       ;
                                     02    A7              15  90 000C5 11$:    MOVB    #21, 2(R7)                                ; 8696
                                           50              56  D0 000C9 12$:    MOVL    VALPTR, R0                                ; 8709
                                                               04 000CC         RET                                               ; 8711
```

; Routine Size: 205 bytes,   Routine Base: DBG$CODE + 23D0

```
;  8607          8712  1 ROUTINE CONVERT_TO_INTEGER (VALPTR) =
;  8608          8713  1 !
;  8609          8714  1 ! FUNCTION
;  8610          8715  1 !       This routine converts a value descriptor to an integer value
;  8611          8716  1 !       and returns the integer value.
;  8612          8717  1 !
;  8613          8718  1 ! INPUTS
;  8614          8719  1 !       VALPTR -        A pointer to a value descriptor
;  8615          8720  1 !
;  8616          8721  1 ! OUTPUTS
;  8617          8722  1 !       An integer value is returned.
;  8618          8723  1 !
;  8619          8724  2     BEGIN
;  8620          8725  2
;  8621          8726  2     LOCAL
;  8622          8727  2         NEW_VALPTR: REF DBG$VALDESC;    ! New value descriptor
;  8623          8728  2
;  8624          8729  2
;  8625          8730  2     ! Build a new value descriptor of type longword integer.
;  8626          8731  2     !
;  8627          8732  2     NEW_VALPTR = DBG$MAKE_SKELETON_DESC(DBG$K_VALUE_DESC, 4);
;  8628          8733  2     NEW_VALPTR[DBG$B_DHDR_KIND] = RST$K_DATA;
;  8629          8734  2     NEW_VALPTR[DBG$B_DHDR_FCODE] = RST$K_TYPE_ATOMIC;
;  8630          8735  2     NEW_VALPTR[DBG$B_VALUE_CLASS] = DSC$K_CLASS_S;
;  8631          8736  2     NEW_VALPTR[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_L;
;  8632          8737  2     NEW_VALPTR[DBG$W_VALUE_LENGTH] = 4;
;  8633          8738  2     NEW_VALPTR[DBG$L_VALUE_POINTER] = NEW_VALPTR[DBG$A_VALUE_ADDRESS];
;  8634          8739  2
;  8635          8740  2
;  8636          8741  2     ! Call the routine which does type conversion.
;  8637          8742  2     !
;  8638          8743  2     NEW_VALPTR = DBG$EVAL_LANG_OPERATOR (
;  8639          8744  2                 DBG$GL_CONVERT_TOKEN, .VALPTR, .NEW_VALPTR);
;  8640          8745  2
;  8641          8746  2     ! Return the value in the new value descriptor.
;  8642          8747  2     !
;  8643          8748  2     RETURN .NEW_VALPTR [DBG$L_VALUE_VALUE0];
;  8644          8749  1     END;


                                0000 00000 CONVERT_TO_INTEGER:
                                           .WORD   Save nothing                                    ; 8712
                                04   DD 00002   PUSHL   #4                                        ; 8732
                          7E         7A   8F   9A 00004   MOVZBL  #122, -(SP)                      ;
00000000G                 00              02   FB 00008   CALLS   #2, DBG$MAKE_SKELETON_DESC      ;
                    06    A0       0602   8F   B0 0000F   MOVW    #1538, 6(NEW_VALPTR)            ; 8734
                    14    A0 01080004   8F   D0 00015   MOVL    #17301508, 20(NEW_VALPTR)       ; 8737
                    18    A0         20   A0   9E 0001D   MOVAB   32(R0), 24(NEW_VALPTR)          ; 8738
                                50   DD 00022   PUSHL   NEW_VALPTR                               ; 8744
                                   04   AC   DD 00024   PUSHL   VALPTR                            ;
                              00000000'  EF   9F 00027   PUSHAB  DBG$GL_CONVERT_TOKEN             ; 8743
00000000G                 00              03   FB 0002D   CALLS   #3, DBG$EVAL_LANG_OPERATOR      ;
                          50         20   A0   D0 00034   MOVL    32(NEW_VALPTR), R0              ; 8748
                                           04   00038   RET                                     ; 8749
```

; Routine Size: 57 bytes,   Routine Base: DBG$CODE + 249D

```
;  8646          8750  1 ROUTINE CREATE_OPERAND_TOKEN(TOKEN_CODE, TOKENBUFFER) =
;  8647          8751  1 !
;  8648          8752  1 ! FUNCTION
;  8649          8753  1 !       This routine creates a Lexical Token Entry for an operand and returns
;  8650          8754  1 !       a pointer to the created Token Entry.  The Token Entry is created in
;  8651          8755  1 !       temporary memory.  When returned, the Lexical Token Entry will have
;  8652          8756  1 !       the TOKEN$B_KIND field set to TOKEN$K_OPERAND and the TOKEN$W_CODE
;  8653          8757  1 !       field set the specified operand token code.  The token name or charac-
;  8654          8758  1 !       ter representation will also be filled into the Token Entry in Counted
;  8655          8759  1 !       ASCII.
;  8656          8760  1 !
;  8657          8761  1 ! INPUTS
;  8658          8762  1 !       TOKEN_CODE - This is the code value which indicates which kind of
;  8659          8763  1 !                    operand this is.  (TOKEN$K_IDENTIFIER or TOKEN$K_STRING
;  8660          8764  1 !                    would be valid examples.)  This value is filled into the
;  8661          8765  1 !                    TOKEN$W_CODE field.
;  8662          8766  1 !
;  8663          8767  1 !       TOKENBUFFER - A pointer to a buffer which contains the Counted ASCII
;  8664          8768  1 !                     representation of the operand for which a Token Entry is to
;  8665          8769  1 !                     be created.  This buffer will thus contain an identifier
;  8666          8770  1 !                     name, the character representation of a numeric constant,
;  8667          8771  1 !                     or the contents o' a character string constant, depending
;  8668          8772  1 !                     on the kind of operand involved.
;  8669          8773  1 !
;  8670          8774  1 ! OUTPUTS
;  8671          8775  1 !       A pointer to a Lexical Token Entry for the specified operand is
;  8672          8776  1 !                     returned as the routine value.
;  8673          8777  1 !
;  8674          8778  1
;  8675          8779  2     BEGIN
;  8676          8780  2
;  8677          8781  2     MAP
;  8678          8782  2         TOKENBUFFER: REF VECTOR[,BYTE]; ! Pointer to token Counted ASCII string
;  8679          8783  2
;  8680          8784  2     LOCAL
;  8681          8785  2         TOKEN: REF TOKEN$ENTRY;         ! Pointer to new Token Entry created
;  8682          8786  2
;  8683          8787  2
;  8684          8788  2
;  8685          8789  2     ! Get a temporary memory block for the Operand Lexical Token Entry.
;  8686          8790  2     ! Fill in the fields of the Token Entry and return its address.
;  8687          8791  2     !
;  8688          8792  2     TOKEN = DBG$GET_TEMPMEM(TOKEN$K_ENTSIZE + .TOKENBUFFER[0]/%UPVAL + 1);
;  8689          8793  2     TOKEN[TOKEN$B_KIND] = TOKEN$K_OPERAND;
;  8690          8794  2     TOKEN[TOKEN$W_CODE] = .TOKEN_CODE;
;  8691          8795  2     CH$MOVE(.TOKENBUFFER[0] + 1, TOKENBUFFER[0], TOKEN[TOKEN$B_LENGTH]);
;  8692          8796  2     RETURN .TOKEN;
;  8693          8797  2
;  8694          8798  1     END;


                                007C 00000 CREATE_OPERAND_TOKEN:
                                           .WORD   Save R2,R3,R4,R5,R6                     ; 8750
                     50    08   BC  9A 00002         MOVZBL  @TOKENBUFFER, R0               ; 8792
```

```
                        50              04  C6 00006       DIVL2   #4, R0                                  ;
                                    03  A0  9F 00009       PUSHAB  3(R0)                                   ;
                00000000G       00      01  FB 0000C       CALLS   #1, DBG$GET_TEMPMEM                     ;
                        56              50  D0 00013       MOVL    R0, TOKEN                               ;
                        66              01  90 00016       MOVB    #1, (TOKEN)                             ; 8793
                    02  A6          04  AC  B0 00019       MOVW    TOKEN_CODE, 2(TOKEN)                    ; 8794
                        50          08  BC  9A 0001E       MOVZBL  @TOKENBUFFER, R0                        ; 8795
                                        50  D6 00022       INCL    R0                                      ;
            08  A6      08  BC          50  28 00024       MOVC3   R0, @TOKENBUFFER, 8(TOKEN)              ;
                        50              56  D0 0002A       MOVL    TOKEN, R0                               ; 8796
                                            04 0002D       RET                                             ; 8798

; Routine Size:  46 bytes,    Routine Base:  DBG$CODE + 24D6
```

```
;  8696          8799  1 ROUTINE CREATE_OPERATOR_TOKEN(TOKEN_CODE, TOKENBUFFER, KIND) =
;  8697          8800  1 !
;  8698          8801  1 ! FUNCTION
;  8699          8802  1 !       This routine creates a Token Entry for an operator and returns a
;  8700          8803  1 !       pointer to the created Token Entry.  The Token Entry is created in
;  8701          8804  1 !       temporary memory.  When returned, the Token Entry will have the
;  8702          8805  1 !       TOKEN$B_KIND field set to the specified kind and the TOKEN$W_CODE
;  8703          8806  1 !       field set the specified operator token code.  The token name or
;  8704          8807  1 !       character representation will also be filled into the Token Entry
;  8705          8808  1 !       in Counted ASCII.
;  8706          8809  1 !
;  8707          8810  1 ! INPUTS
;  8708          8811  1 !       TOKEN_CODE - This is the code value which indicates which type of
;  8709          8812  1 !                    operator this is.  (TOKEN$K_DOT, TOKEN$K_ADD, TOKEN$K_BIF_OP
;  8710          8813  1 !                    would be valid examples.)  This value is filled into the
;  8711          8814  1 !                    TOKEN$W_CODE field.
;  8712          8815  1 !
;  8713          8816  1 !       TOKENBUFFER - A pointer to a buffer which contains the Counted ASCII
;  8714          8817  1 !                     representation of the operator for which a Token Entry is to
;  8715          8818  1 !                     be created.  This buffer will thus contain an identifier
;  8716          8819  1 !                     name, or the contents of a character string constant,
;  8717          8820  1 !                     depending on the kind of operator involved.
;  8718          8821  1 !
;  8719          8822  1 !       KIND - This is the code value which indicates which kind of operator
;  8720          8823  1 !              this is.  (e.g. TOKEN$K_PREFIX_OP)  This value is filled
;  8721          8824  1 !              into the TOKEN$B_KIND field.
;  8722          8825  1 !
;  8723          8826  1 ! OUTPUTS
;  8724          8827  1 !       A pointer to a Token Entry for the specified operand is returned as
;  8725          8828  1 !       the routine value.
;  8726          8829  1 !
;  8727          8830  1
;  8728          8831  2     BEGIN
;  8729          8832  2
;  8730          8833  2     MAP
;  8731          8834  2         TOKENBUFFER: REF VECTOR[,BYTE]; ! Pointer to token Counted ASCII string
;  8732          8835  2
;  8733          8836  2     LOCAL
;  8734          8837  2         TOKEN: REF TOKEN$ENTRY;         ! Pointer to new Token Entry created
;  8735          8838  2
;  8736          8839  2
;  8737          8840  2
;  8738          8841  2     ! Get a temporary memory block for the Operator Token Entry.
;  8739          8842  2     ! Fill in the fields of the Token Entry and return its address.
;  8740          8843  2     !
;  8741          8844  2     TOKEN = DBG$GET_TEMPMEM(TOKEN$K_ENTSIZE_OPERATOR + .TOKENBUFFER[0]/%UPVAL + 1);
;  8742          8845  2     TOKEN[TOKEN$B_KIND] = .KIND;
;  8743          8846  2     TOKEN[TOKEN$W_CODE] = .TOKEN_CODE;
;  8744          8847  2     CH$MOVE(.TOKENBUFFER[0] + 1, TOKENBUFFER[0], TOKEN[TOKEN$B_OPLEN]);
;  8745          8848  2     RETURN .TOKEN;
;  8746          8849  2
;  8747          8850  1     END;
```

```
                                        007C 00000 CREATE_OPERATOR_TOKEN:
                                                           .WORD   Save R2,R3,R4,R5,R6                                     ; 8799
                                 50      08   BC  9A 00002         MOVZBL  @TOKENBUFFER, R0                                ; 8844
                                 50           04  C6 00006         DIVL2   #4, R0                                          ;
                                              04  A0 00009         PUSHAB  4(R0)                                           ;
                        00000000G 00          01  FB 0000C         CALLS   #1, DBG$GET_TEMPMEM                             ;
                                 56           50  D0 00013         MOVL    R0, TOKEN                                       ;
                                 66      0C   AC  90 00016         MOVB    KIND, (TOKEN)                                   ; 8845
                           02    A6      04   AC  B0 0001A         MOVW    TOKEN_CODE, 2(TOKEN)                            ; 8846
                                 50      08   BC  9A 0001F         MOVZBL  @TOKENBUFFER, R0                                ; 8847
                                              50  D6 00023         INCL    R0                                              ;
                     0C    A6         08   BC      50  28 00025   MOVC3   R0, @TOKENBUFFER, 12(TOKEN)                     ;
                                 50           56  D0 0002B         MOVL    TOKEN, R0                                       ; 8848
                                              04  0002E         RET                                                     ; 8850

; Routine Size:  47 bytes,    Routine Base:  DBG$CODE + 2504
```

```
;  8749          8851  1 ROUTINE CREATE_PRID_CONSTANT(PRID) =
;  8750          8852  1 !
;  8751          8853  1 ! FUNCTION
;  8752          8854  1 !       This routine creates a value descriptor for the predefined identifier
;  8753          8855  1 !       constant and returns the pointer to the value descriptor.
;  8754          8856  1 !
;  8755          8857  1 ! INPUTS
;  8756          8858  1 !       PRID    - Pointer to a Predefined Identifier Constant entry in
;  8757          8859  1 !                 Predefined Identifier Table for the given language.
;  8758          8860  1 !
;  8759          8861  1 ! OUTPUTS
;  8760          8862  1 !       A pointer to the value descriptor of Predefined Identifier Constant
;  8761          8863  1 !       is returned.
;  8762          8864  1 !
;  8763          8865  1
;  8764          8866  2     BEGIN
;  8765          8867  2
;  8766          8868  2     MAP
;  8767          8869  2         PRID: REF PRID$ENTRY;           ! Pointer to Predefined Identifier
;  8768          8870  2
;  8769          8871  2     LOCAL
;  8770          8872  2         VALPTR: REF DBG$VALDESC;        ! Pointer to Value Descriptor
;  8771          8873  2
;  8772          8874  2
;  8773          8875  2     VALPTR = DBG$MAKE_SKELETON_DESC(DBG$K_VALUE_DESC, 4);
;  8774          8876  2     VALPTR[DBG$B_DHDR_LANG] = .DBG$GB_LANGUAGE;
;  8775          8877  2     VALPTR[DBG$B_DHDR_KIND] = RST$K_DATA;
;  8776          8878  2     VALPTR[DBG$B_DHDR_FCODE] = .PRID[PRID$B_FCODE];
;  8777          8879  2     VALPTR[DBG$B_VALUE_DTYPE] = .PRID[PRID$B_DTYPE];
;  8778          8880  2     VALPTR[DBG$B_VALUE_CLASS] = DBG$MAP_DTYPE_CLASS(
;  8779          8881  2                                 .PRID[PRID$B_DTYPE], FALSE);
;  8780          8882  2     VALPTR[DBG$W_VALUE_LENGTH] = DBG$NUM_BYTES(.PRID[PRID$B_DTYPE]);
;  8781          8883  2     VALPTR[DBG$L_VALUE_POINTER] = VALPTR[DBG$A_VALUE_ADDRESS];
;  8782          8884  2     VALPTR[DBG$L_VALUE_VALUE0] = .PRID[PRID$L_VALUE];
;  8783          8885  2     RETURN .VALPTR;
;  8784          8886  1     END;


                                000C 00000 CREATE_PRID_CONSTANT:
                                           .WORD   Save R2,R3                          ;  8851
                                04   DD 00002 PUSHL  #4                               ;  8875
                          7E        7A   8F   9A 00004 MOVZBL #122, -(SP)             ;
00000000G                 00             02   FB 00008 CALLS  #2, DBG$MAKE_SKELETON_DESC ;
                          52             50   D0 0000F MOVL   R0, VALPTR             ;
                03        A2 00000000G   00   90 00012 MOVB   DBG$GB_LANGUAGE, 3(VALPTR) ;  8876
                07        A2             06   90 0001A MOVB   #6, 7(VALPTR)          ;  8877
                          53        04   AC   D0 0001E MOVL   PRID, R3               ;  8878
                06        A2        02   A3   90 00022 MOVB   2(R3), 6(VALPTR)       ;
                16        A2        01   A3   90 00027 MOVB   1(R3), 22(VALPTR)      ;  8879
                                    7E   D4 0002C CLRL   -(SP)                       ;  8880
                          7E        01   A3   9A 0002E MOVZBL 1(R3), -(SP)          ;  8881
00000000G                 00             02   FB 00032 CALLS  #2, DBG$MAP_DTYPE_CLASS ;
                17        A2             50   90 00039 MOVB   R0, 23(VALPTR)       ;
                          7E        01   A3   9A 0003D MOVZBL 1(R3), -(SP)          ;  8882
```

```
                00000000G  00          01  FB 00041        CALLS   #1, DBG$NUM_BYTES                       ;
                      14   A2          50  B0 00048        MOVW    R0, 20(VALPTR)                          ;
                      18   A2    20    A2  9E 0004C        MOVAB   32(VALPTR), 24(VALPTR)                  ; 8883
                      20   A2    04    A3  D0 00051        MOVL    4(R3), 32(VALPTR)                       ; 8884
                           50          52  D0 00056        MOVL    VALPTR, R0                              ; 8885
                                       04 00059        RET                                             ; 8886

; Routine Size:  90 bytes,    Routine Base:  DBG$CODE + 2533
```

```
;  8786          8887  1 ROUTINE DUMP_OPERATOR(OPERATOR, PRIMARY_FLAG): NOVALUE =
;  8787          8888  1 !
;  8788          8889  1 ! FUNCTION
;  8789          8890  1 !       This routine dumps out an Operator Lexical Token Entry in a readable
;  8790          8891  1 !       format when that operator is about to be evaluated.  It prints one
;  8791          8892  1 !       line of output of approximately this format:
;  8792          8893  1 !
;  8793          8894  1 !                       primary operator "\" evaluated (infix)
;  8794          8895  1 !
;  8795          8896  1 !       This routine is used only for internal DEBUG debugging purposes.  Its
;  8796          8897  1 !       output is not seen by normal DEBUG users.  It is only called when
;  8797          8898  1 !       Developer Switch 3 is set to trace operator evaluations as they occur.
;  8798          8899  1 !
;  8799          8900  1 ! INPUTS
;  8800          8901  1 !       OPERATOR - A pointer to the Lexical Token Entry for the operator
;  8801          8902  1 !                  to be dumped.
;  8802          8903  1 !
;  8803          8904  1 !       PRIMARY_FLAG - A flag whose value is TRUE if this operator is
;  8804          8905  1 !                  evaluated as part of a Primary Symbol.  Its value is
;  8805          8906  1 !                  FALSE if it is an ordinary expression operator.
;  8806          8907  1 !
;  8807          8908  1 ! OUTPUTS
;  8808          8909  1 !       NONE
;  8809          8910  1 !
;  8810          8911  1
;  8811          8912  2     BEGIN
;  8812          8913  2
;  8813          8914  2     MAP
;  8814          8915  2         OPERATOR: REF TOKEN$ENTRY;      ! Pointer to operator's token entry
;  8815          8916  2
;  8816          8917  2
;  8817          8918  2
;  8818          8919  2     ! Print the main text of the message including the operator name string.
;  8819          8920  2     !
;  8820          8921  2     DBG$PRINT(UPLIT BYTE(%ASCIC '    '), 0);
;  8821          8922  2     IF .PRIMARY_FLAG THEN DBG$PRINT(UPLIT BYTE(%ASCIC 'primary '), 0);
;  8822          8923  2     DBG$PRINT(UPLIT BYTE(%ASCIC 'operator "!AC" evaluated ('),
;  8823          8924  2                                             OPERATOR[TOKEN$B_OPLEN]);
;  8824          8925  2
;  8825          8926  2
;  8826          8927  2     ! Then print what kind of operator this is.
;  8827          8928  2     !
;  8828          8929  2     IF .OPERATOR[TOKEN$B_KIND] EQL TOKEN$K_PREFIX_OP
;  8829          8930  2     THEN
;  8830          8931  2         DBG$PRINT(UPLIT BYTE(%ASCIC 'prefix'), 0)
;  8831          8932  2
;  8832          8933  2     ELSE IF .OPERATOR[TOKEN$B_KIND] EQL TOKEN$K_INFIX_OP
;  8833          8934  2     THEN
;  8834          8935  2         DBG$PRINT(UPLIT BYTE(%ASCIC 'infix'), 0)
;  8835          8936  2
;  8836          8937  2     ELSE IF .OPERATOR[TOKEN$B_KIND] EQL TOKEN$K_POSTFIX_OP
;  8837          8938  2     THEN
;  8838          8939  2         DBG$PRINT(UPLIT BYTE(%ASCIC 'postfix'), 0)
;  8839          8940  2
;  8840          8941  2     ELSE
;  8841          8942  2         DBG$PRINT(UPLIT BYTE(%ASCIC 'invalid kind'), 0);
;  8842          8943  2
```

```
;  8843          8944  2
;  8844          8945  2     ! Close out the message, flush the buffer, and return.
;  8845          8946  2     !
;  8846          8947  2     DBG$PRINT(UPLIT BYTE(%ASCIC ')'), 0);
;  8847          8948  2     DBG$NEWLINE();
;  8848          8949  2     RETURN;
;  8849          8950  2
;  8850          8951  1     END;


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

                                                20  20  20  20  04  03219 P.AXN:  .ASCII  <4>\    \                    ;
                                            20  79  72  61  6D  69  72  70  08  0321E P.AXO:  .ASCII  <8>\primary \                ;
22  43  41  21  22  20  72  6F  74  61  72  65  70  6F  1A  03227 P.AXP:  .ASCII  <26>\operator "'!AC" evaluated (\          ;
                28  20  64  65  74  61  75  6C  61  76  65  20  03236                                                          ;
                                        78  69  66  65  72  70  06  03242 P.AXQ:  .ASCII  <6>\prefix\                         ;
                                        78  69  66  6E  69  05  03249 P.AXR:  .ASCII  <5>\infix\                           ;
                                    78  69  66  74  73  6F  70  07  0324F P.AXS:  .ASCII  <7>\postfix\                         ;
            64  6E  69  6B  20  64  69  6C  61  76  6E  69  0C  03257 P.AXT:  .ASCII  <12>\invalid kind\                      ;
                                                        29  01  03264 P.AXU:  .ASCII  <1>\)\                                ;


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

                                                000C 00000 DUMP_OPERATOR:
                                                            .WORD   Save R2,R3                                          ; 8887
                    53 00000000G    00  9E 00002            MOVAB   DBG$PRINT, R3
                    52 00000000'    EF  9E 00009            MOVAB   P.AXN, R2
                                    7E  D4 00010            CLRL    -(SP)                                               ; 8921
                                    52  DD 00012            PUSHL   R2
                    63              02  FB 00014            CALLS   #2, DBG$PRINT
                    08      08      AC  E9 00017            BLBC    PRIMARY_FLAG, 1$                                    ; 8922
                                    7E  D4 0001B            CLRL    -(SP)
                            05      A2  9F 0001D            PUSHAB  P.AXO
                    63              02  FB 00020            CALLS   #2, DBG$PRINT
7E          04      AC              0C  C1 00023 1$:        ADDL3   #12, OPERATOR, -(SP)                               ; 8924
                            0E      A2  9F 00028            PUSHAB  P.AXP                                               ; 8923
                    63              02  FB 0002B            CALLS   #2, DBG$PRINT                                       ; 8924
                    02      04      BC  91 0002E            CMPB    @OPERATOR, #2                                       ; 8929
                                    07  12 00032            BNEQ    2$
                                    7E  D4 00034            CLRL    -(SP)                                               ; 8931
                            29      A2  9F 00036            PUSHAB  P.AXQ
                                    1F  11 00039            BRB     5$
                    03      04      BC  91 0003B 2$:        CMPB    @OPERATOR, #3                                       ; 8933
                                    07  12 0003F            BNEQ    3$
                                    7E  D4 00041            CLRL    -(SP)                                               ; 8935
                            30      A2  9F 00043            PUSHAB  P.AXR
                                    12  11 00046            BRB     5$
                    04      04      BC  91 00048 3$:        CMPB    @OPERATOR, #4                                       ; 8937
                                    07  12 0004C            BNEQ    4$
                                    7E  D4 0004E            CLRL    -(SP)                                               ; 8939
                            36      A2  9F 00050            PUSHAB  P.AXS
                                    05  11 00053            BRB     5$
                                    7E  D4 00055 4$:        CLRL    -(SP)                                               ; 8942
```

```
                          3E   A2  9F 00057          PUSHAB  P.AXT                                    ;
                     63        02  FB 0005A 5$:      CALLS   #2, DBG$PRINT                           ;
                               7E  D4 0005D          CLRL    -(SP)                                   ; 8947
                          4B   A2  9F 0005F          PUSHAB  P.AXU                                   ;
                     63        02  FB 00062          CALLS   #2, DBG$PRINT                           ;
           00000000G 00        00  FB 00065          CALLS   #0, DBG$NEWLINE                         ; 8948
                               04  0006C          RET                                             ; 8951

; Routine Size:  109 bytes,    Routine Base:  DBG$CODE + 258D
```

```
;  8852          8952  1 ROUTINE DUMP_TOKEN(TOKEN): NOVALUE =
;  8853          8953  1 !
;  8854          8954  1 ! FUNCTION
;  8855          8955  1 !       This routine dumps out a specified Lexical Token Entry in a readable
;  8856          8956  1 !       format.  It prints one line of output of approximately this format:
;  8857          8957  1 !
;  8858          8958  1 !               token found: infix operator, code = 8, string = "**"
;  8859          8959  1 !
;  8860          8960  1 !       This routine is used only for internal DEBUG debugging purposes.
;  8861          8961  1 !       Its output is not seen by normal DEBUG users.  The routine is only
;  8862          8962  1 !       called by DBG$PRIMARY_PARSER if Developer Switch 2 is set.
;  8863          8963  1 !
;  8864          8964  1 ! INPUTS
;  8865          8965  1 !       TOKEN   - The address of the Lexical Token Entry to dump.
;  8866          8966  1 !
;  8867          8967  1 ! OUTPUTS
;  8868          8968  1 !       NONE
;  8869          8969  1 !
;  8870          8970  1
;  8871          8971  2     BEGIN
;  8872          8972  2
;  8873          8973  2     MAP
;  8874          8974  2         TOKEN: REF TOKEN$ENTRY;         ! Address of Lexical Token Entry to dump
;  8875          8975  2
;  8876          8976  2     LOCAL
;  8877          8977  2         KIND,                           ! Pointer to text saying what kind of
;  8878          8978  2                                         !       lexical token this is
;  8879          8979  2         NAMEPTR,                        ! Pointer to the ASCIC name string for
;  8880          8980  2                                         !       the lexical token
;  8881          8981  2         PRIMARY;                        ! Pointer to string saying whether this
;  8882          8982  2                                         !       is a Primary Symbol operator
;  8883          8983  2
;  8884          8984  2
;  8885          8985  2
;  8886          8986  2     ! Determine what kind of Lexical Token Entry this is.
;  8887          8987  2     !
;  8888          8988  2     CASE .TOKEN[TOKEN$B_KIND] FROM TOKEN$K_OPERAND TO TOKEN$K_POSTFIX_OP OF
;  8889          8989  2         SET
;  8890          8990  2
;  8891          8991  2
;  8892          8992  2         [TOKEN$K_OPERAND]:
;  8893          8993  3             BEGIN
;  8894          8994  3             KIND = UPLIT BYTE(%ASCIC 'operand');
;  8895          8995  3             NAMEPTR = TOKEN[TOKEN$B_LENGTH];
;  8896          8996  2             END;
;  8897          8997  2
;  8898          8998  2
;  8899          8999  2         [TOKEN$K_PREFIX_OP]:
;  8900          9000  3             BEGIN
;  8901          9001  3             KIND = UPLIT BYTE(%ASCIC 'prefix operator');
;  8902          9002  3             NAMEPTR = TOKEN[TOKEN$B_OPLEN];
;  8903          9003  2             END;
;  8904          9004  2
;  8905          9005  2
;  8906          9006  2         [TOKEN$K_INFIX_OP]:
;  8907          9007  3             BEGIN
;  8908          9008  3             KIND = UPLIT BYTE(%ASCIC 'infix operator');
```

```
;  8909          9009  3              NAMEPTR = TOKEN[TOKEN$B_OPLEN];
;  8910          9010  2              END;
;  8911          9011  2
;  8912          9012  2
;  8913          9013  2          [TOKEN$K_POSTFIX_OP]:
;  8914          9014  3              BEGIN
;  8915          9015  3              KIND = UPLIT BYTE(%ASCIC 'postfix operator');
;  8916          9016  3              NAMEPTR = TOKEN[TOKEN$B_OPLEN];
;  8917          9017  2              END;
;  8918          9018  2
;  8919          9019  2
;  8920          9020  2          [INRANGE, OUTRANGE]:
;  8921          9021  3              BEGIN
;  8922          9022  3              KIND = UPLIT BYTE(%ASCIC 'invalid kind');
;  8923          9023  3              NAMEPTR = UPLIT BYTE(%ASCIC '');
;  8924          9024  2              END;
;  8925          9025  2
;  8926          9026  2          TES;
;  8927          9027  2
;  8928          9028  2
;  8929          9029  2      ! Print the line describing the Lexical Token Entry.
;  8930          9030  2      !
;  8931          9031  2      PRIMARY = UPLIT BYTE(%ASCIC '');
;  8932          9032  2      IF .TOKEN[TOKEN$V_PRIMARY] THEN PRIMARY = UPLIT BYTE(%ASCIC ' (primary)');
;  8933          9033  2      DBG$PRINT(UPLIT BYTE(%ASCIC 'token found: !AC!AC, code = !SL, string = ''!AC'''),
;  8934          9034  2                .KIND, .PRIMARY, .TOKEN[TOKEN$W_CODE], .NAMEPTR);
;  8935          9035  2      DBG$NEWLINE();
;  8936          9036  2      RETURN;
;  8937          9037  2
;  8938          9038  1      END;


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

                                  64 6E 61 72 65 70 6F 07 03266 P.AXV:  .ASCII  <7>\operand\                          ;
6F 74 61 72 65 70 6F 20 78 69 66 65 72 70 0F 0326E P.AXW:  .ASCII  <15>\prefix operator\                     ;
                                                  72 0327D                                                  ;
72 6F 74 61 72 65 70 6F 20 78 69 66 6E 69 0E 0327E P.AXX:  .ASCII  <14>\infix operator\                       ;
74 61 72 65 70 6F 20 78 69 66 74 73 6F 70 10 0328D P.AXY:  .ASCII  <16>\postfix operator\                     ;
                                                  72 6F 0329C                                               ;
      64 6E 69 6B 20 64 69 6C 61 76 6E 69 0C 0329E P.AXZ:  .ASCII  <12>\invalid kind\                         ;
                                                  00 032AB P.AYA:  .ASCII  <0>                              ;
                                                  00 032AC P.AYB:  .ASCII  <0>                              ;
            29 79 72 61 6D 69 72 70 28 20 0A 032AD P.AYC:  .ASCII  <10>\ (primary)\                         ;
21 20 3A 64 6E 75 6F 66 20 6E 65 6B 6F 74 2F 032B8 P.AYD:  .ASCII  \/token found: !AC!AC, code = !SL, string\ ;
21 20 3D 20 65 64 6F 63 20 2C 43 41 21 43 41 032C7                                                          ;
               67 6E 69 72 74 73 20 2C 4C 53 032D6                                                          ;
                  22 43 41 21 22 20 3D 20 032E0          .ASCII  \ = ''!AC''\                               ;


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

                                                001C 00000 DUMP_TOKEN:
                                                           .WORD   Save R2,R3,R4                            ; 8952
                                  54 00000000'  EF  9E 00002       MOVAB   P.AXZ, R4                         ;
```

```
                                   51      04  AC  D0 00009        MOVL    TOKEN, R1                        ; 8988
                         03        01          61  8F 0000D        CASEB   (R1), #1, #3                     ;
        0027           0021      001B        0011     00011 1$:    .WORD   2$-1$,-                          ;
                                                                           3$-1$,-                          ;
                                                                           4$-1$,-                          ;
                                                                           5$-1$                            ;
                                   53          64  9E 00019        MOVAB   P.AXZ, KIND                      ; 9022
                                   50      0D  A4  9E 0001C        MOVAB   P.AYA, NAMEPTR                   ; 9023
                                               1E  11 00020        BRB     7$                               ; 8988
                                   53      C8  A4  9E 00022 2$:    MOVAB   P.AXV, KIND                      ; 8994
                                   50      08  A1  9E 00026        MOVAB   8(R1), NAMEPTR                   ; 8995
                                               14  11 0002A        BRB     7$                               ; 8988
                                   53      D0  A4  9E 0002C 3$:    MOVAB   P.AXW, KIND                      ; 9001
                                               0A  11 00030        BRB     6$                               ; 9002
                                   53      E0  A4  9E 00032 4$:    MOVAB   P.AXX, KIND                      ; 9008
                                               04  11 00036        BRB     6$                               ; 9009
                                   53      EF  A4  9E 00038 5$:    MOVAB   P.AXY, KIND                      ; 9015
                                   50      0C  A1  9E 0003C 6$:    MOVAB   12(R1), NAMEPTR                  ; 9016
                                   52      0E  A4  9E 00040 7$:    MOVAB   P.AYB, PRIMARY                   ; 9031
                                   04      01  A1  E9 00044        BLBC    1(R1), 8$                        ; 9032
                                   52      0F  A4  9E 00048        MOVAB   P.AYC, PRIMARY                   ;
                                               50  DD 0004C 8$:    PUSHL   NAMEPTR                          ; 9034
                                   7E      02  A1  3C 0004E        MOVZWL  2(R1), -(SP)                     ;
                                               52  DD 00052        PUSHL   PRIMARY                          ;
                                               53  DD 00054        PUSHL   KIND                             ;
                                           1A  A4  9F 00056        PUSHAB  P.AYD                            ; 9033
                       00000000G  00           05  FB 00059        CALLS   #5, DBG$PRINT                    ;
                       00000000G  00           00  FB 00060        CALLS   #0, DBG$NEWLINE                  ; 9035
                                                   04 00067        RET                                      ; 9038

; Routine Size:  104 bytes,    Routine Base:  DBG$CODE + 25FA
```

```
;  8940          9039  1 ROUTINE DUMP_PRIMARY(PRIMPTR): NOVALUE =
;  8941          9040  1 !
;  8942          9041  1 ! FUNCTION
;  8943          9042  1 !       This routine dumps out a Primary Descriptor or a Value Descriptor
;  8944          9043  1 !       in hexadecimal on the terminal.  For a Primary Descriptor, it dumps
;  8945          9044  1 !       out not only the Root Node but also all Sub-Nodes.  This routine is
;  8946          9045  1 !       used only for internal DEBUG debugging purposes.  Its output is not
;  8947          9046  1 !       seen by normal DEBUG users.  It is only called if Developer Switch
;  8948          9047  1 !       3 is set.
;  8949          9048  1 !
;  8950          9049  1 ! INPUTS
;  8951          9050  1 !       PRIMPTR - A pointer to the Primary Descriptor or the Value Descriptor
;  8952          9051  1 !                 to be dumped out.
;  8953          9052  1 !
;  8954          9053  1 ! OUTPUTS
;  8955          9054  1 !       NONE
;  8956          9055  1 !
;  8957          9056  1
;  8958          9057  2     BEGIN
;  8959          9058  2
;  8960          9059  2     MAP
;  8961          9060  2         PRIMPTR: REF DBG$PRIMARY;       ! Pointer to Primary Descriptor
;  8962          9061  2
;  8963          9062  2     LOCAL
;  8964          9063  2         LENGTH,                         ! Byte length of Primary Descr Sub-Node
;  8965          9064  2         NODEPTR: REF DBG$PRIM_NODE;     ! Pointer to Primary Descr Sub-Node
;  8966          9065  2
;  8967          9066  2
;  8968          9067  2
;  8969          9068  2     ! If this is a Primary Descriptor, dump out the Root Node and then loop to
;  8970          9069  2     ! dump out all the individual Sub-Nodes.
;  8971          9070  2     !
;  8972          9071  2     IF .PRIMPTR[DBG$B_DHDR_TYPE] EQL DBG$K_PRIMARY_DESC
;  8973          9072  2     THEN
;  8974          9073  3         BEGIN
;  8975          9074  3
;  8976          9075  3
;  8977          9076  3         ! Dump out the Primary Descriptor Root Node.
;  8978          9077  3         !
;  8979          9078  3         DBG$DUMP_HEX(.PRIMPTR, DBG$K_PRIMARY_SIZE*%UPVAL, .PRIMPTR,
;  8980          9079  3                     UPLIT BYTE (%ASCIC '      Primary Descriptor:'));
;  8981          9080  3
;  8982          9081  3
;  8983          9082  3         ! Dump out each of the Primary Descriptor Sub-Nodes.  Note that the
;  8984          9083  3         ! length of the Sub-Node in bytes depends on the FCODE stored in the
;  8985          9084  3         ! Sub-Node.
;  8986          9085  3         !
;  8987          9086  3         NODEPTR = .PRIMPTR[DBG$L_PRIM_FLINK];
;  8988          9087  3         WHILE .NODEPTR NEQ PRIMPTR[DBG$A_PRIM_FLINK] DO
;  8989          9088  4             BEGIN
;  8990          9089  4             IF .NODEPTR[DBG$B_PNODE_FCODE] EQL RST$K_TYPE_ARRAY
;  8991          9090  4             THEN
;  8992          9091  5                 LENGTH = %UPVAL*(DBG$K_PRIM_SIZE_ARRAY +
;  8993          9092  5                         DBG$K_PRIM_SIZE_SUBS*.NODEPTR[DBG$B_PNARR_DIMCNT])
;  8994          9093  5
;  8995          9094  4             ELSE IF .NODEPTR[DBG$B_PNODE_FCODE] EQL RST$K_TYPE_RECORD
;  8996          9095  4             THEN
```

```
;  8997          9096  4                 LENGTH = DBG$K_PRIM_SIZE_RECORD*%UPVAL
;  8998          9097  4
;  8999          9098  4             ELSE IF .NODEPTR[DBG$B_PNODE_FCODE] EQL RST$K_TYPE_VARIANT
;  9000          9099  4             THEN
;  9001          9100  4                 LENGTH = DBG$K_PRIM_SIZE_VARIANT*%UPVAL
;  9002          9101  4
;  9003          9102  4             ELSE
;  9004          9103  4                 LENGTH = DBG$K_PRIM_SIZE_NORMAL*%UPVAL;
;  9005          9104  4
;  9006          9105  4             DBG$DUMP_HEX(.NODEPTR, .LENGTH, .NODEPTR,
;  9007          9106  4                          UPLIT BYTE (%ASCIC '      Primary Sub-Node:'));
;  9008          9107  4             NODEPTR = .NODEPTR[DBG$L_PNODE_FLINK];
;  9009          9108  3             END;
;  9010          9109  3
;  9011          9110  3         END
;  9012          9111  3
;  9013          9112  3
;  9014          9113  3     ! If this is a Value Descriptor, dump it as such.
;  9015          9114  3     !
;  9016          9115  2     ELSE IF .PRIMPTR[DBG$B_DHDR_TYPE] EQL DBG$K_VALUE_DESC
;  9017          9116  2     THEN
;  9018          9117  2         DBG$DUMP_HEX(.PRIMPTR, .PRIMPTR[DBG$W_DHDR_LENGTH], .PRIMPTR,
;  9019          9118  2                      UPLIT BYTE (%ASCIC '      Value Descriptor:'))
;  9020          9119  2
;  9021          9120  2
;  9022          9121  2     ! If this is a Volatile Value Descriptor, dump it as such.
;  9023          9122  2     !
;  9024          9123  2     ELSE IF .PRIMPTR[DBG$B_DHDR_TYPE] EQL DBG$K_V_VALUE_DESC
;  9025          9124  2     THEN
;  9026          9125  2         DBG$DUMP_HEX(.PRIMPTR, .PRIMPTR[DBG$W_DHDR_LENGTH]  .PRIMPTR,
;  9027          9126  2                      UPLIT BYTE (%ASCIC '      Volatile Value Descriptor:'))
;  9028          9127  2
;  9029          9128  2
;  9030          9129  2     ! If it is none of the above, something is wrong but we try to dump the
;  9031          9130  2     ! descriptor anyway.
;  9032          9131  2     !
;  9033          9132  2     ELSE
;  9034          9133  2         DBG$DUMP_HEX(.PRIMPTR, .PRIMPTR[DBG$W_DHDR_LENGTH], .PRIMPTR,
;  9035          9134  2                      UPLIT BYTE (%ASCIC '      Invalid Descriptor Type:'));
;  9036          9135  2
;  9037          9136  2
;  9038          9137  2     ! We are all done--now return.
;  9039          9138  2     !
;  9040          9139  2     RETURN;
;  9041          9140  1     END;


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

20 79 72 61 6D 69 72 50 20 20 20 20 20 20 20 19  032E8 P.AYE:   .ASCII  <25>\      Primary Descriptor:\          ;
                3A 72 6F 74 70 69 72 63 73 65 44  032F7
20 79 72 61 6D 69 72 50 20 20 20 20 20 20 20 17  03302 P.AYF:   .ASCII  <23>\      Primary Sub-Node:\            ;
                   3A 65 64 6F 4E 2D 62 75 53     03311
65 44 20 65 75 6C 61 56 20 20 20 20 20 20 20 17  0331A P.AYG:   .ASCII  <23>\      Value Descriptor:\            ;
                   3A 72 6F 74 70 69 72 63 73     03329
65 6C 69 74 61 6C 6F 56 20 20 20 20 20 20 20 20  03332 P.AYH:   .ASCII  \       Volatile Value Descriptor:\   ;
```

```
74 70 69 72 63 73 65 44 20 65 75 6C 61 56 20  03341
                              3A 72 6F        03350
20 64 69 6C 61 76 6E 49 20 20 20 20 20 20 1E  03353 P.AYI:  .ASCII  <30>\      Invalid Descriptor Type:\
65 70 79 54 20 72 6F 74 70 69 72 63 73 65 44  03362
                                       3A     03371


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

                                          007C 00000 DUMP_PRIMARY:
                                                        .WORD   Save R2,R3,R4,R5,R6                 ; 9039
                    56 00000000G    00  9E 00002          MOVAB   DBG$DUMP_HEX, R6
                    55 00000000'    EF  9E 00009          MOVAB   P.AYE, R5
                    54        04    AC  D0 00010          MOVL    PRIMPTR, R4                       ; 9071
                 79 8F        02    A4  91 00014          CMPB    2(R4), #121
                                    51  12 00019          BNEQ    6$
                                    30  BB 0001B          PUSHR   #^M<R4,R5>                        ; 9078
                                    24  DD 0001D          PUSHL   #36
                                    54  DD 0001F          PUSHL   R4
                    66              04  FB 00021          CALLS   #4, DBG$DUMP_HEX
                    52        14    A4  D0 00024          MOVL    20(R4), NODEPTR                   ; 9086
                    50        14    A4  9E 00028 1$:      MOVAB   20(R4), R0                        ; 9087
                    50              52  D1 0002C          CMPL    NODEPTR, R0
                                    60  13 0002F          BEQL    10$
                    01        09    A2  91 00031          CMPB    9(NODEPTR), #1                    ; 9089
                                    0D  12 00035          BNEQ    2$
                    50        1B    A2  9A 00037          MOVZBL  27(NODEPTR), R0                   ; 9092
                    50              14  C4 0003B          MULL2   #20, R0
                    53        28    A0  9E 0003E          MOVAB   40(R0), LENGTH                    ; 9091
                                    19  11 00042          BRB     5$
                    07        09    A2  91 00044 2$:      CMPB    9(NODEPTR), #7                    ; 9094
                                    05  12 00048          BNEQ    3$
                    53              1C  D0 0004A          MOVL    #28, LENGTH                       ; 9096
                                    0E  11 0004D          BRB     5$
                    13        09    A2  91 0004F 3$:      CMPB    9(NODEPTR), #19                   ; 9098
                                    05  12 00053          BNEQ    4$
                    53              28  D0 00055          MOVL    #40, LENGTH                       ; 9100
                                    03  11 00058          BRB     5$
                    53              18  D0 0005A 4$:      MOVL    #24, LENGTH                       ; 9103
                              1A    A5  9F 0005D 5$:      PUSHAB  P.AYF                             ; 9106
                                    52  DD 00060          PUSHL   NODEPTR                           ; 9105
                                    0C  BB 00062          PUSHR   #^M<R2,R3>
                    66              04  FB 00064          CALLS   #4, DBG$DUMP_HEX
                    52              62  D0 00067          MOVL    (NODEPTR), NODEPTR                ; 9107
                                    BC  11 0006A          BRB     1$                                ; 9087
                 7A 8F        02    A4  91 0006C 6$:      CMPB    2(R4), #122                       ; 9115
                                    05  12 00071          BNEQ    7$
                              32    A5  9F 00073          PUSHAB  P.AYG                             ; 9118
                                    0F  11 00076          BRB     9$                                ; 9117
                 83 8F        02    A4  91 00078 7$:      CMPB    2(R4), #131                       ; 9123
                                    05  12 0007D          BNEQ    8$
                              4A    A5  9F 0007F          PUSHAB  P.AYH                             ; 9126
                                    03  11 00082          BRB     9$                                ; 9125
                              6B    A5  9F 00084 8$:      PUSHAB  P.AYI                             ; 9134
                                    54  DD 00087 9$:      PUSHL   R4                                ; 9133
                    7E              64  3C 00089          MOVZWL  (R4), -(SP)
```

```
                                54  DD 0008C           PUSHL   R4                                                      ;
                             66 04  FB 0008E           CALLS   #4, DBG$DUMP_HEX                                        ;
                                    04 00091 10$:      RET                                                             ; 9140

; Routine Size:  146 bytes,    Routine Base:  DBG$CODE + 2662
```

```
;  9043          9141  1 ROUTINE FIX_UP_PRIMARY(PRIMPTR): NOVALUE =
;  9044          9142  1 !
;  9045          9143  1 ! FUNCTION
;  9046          9144  1 !       This routine is needed to handle a special case that may arise
;  9047          9145  1 !       in conjunction with the array slice feature: If X is a two -
;  9048          9146  1 !       dimensional array, say 10x10, and the user specifies EX X[5],
;  9049          9147  1 !       then we want to treat this as if he had said X[5][1:10].
;  9050          9148  1 !       However, we do not know at the time we are picking up the "5"
;  9051          9149  1 !       subscript whether he is going to say X[5] or X[5][6], for example.
;  9052          9150  1 !       So we cannot fix up the lower/upper bounds properly until after
;  9053          9151  1 !       all the subscripts have been picked up. This is the routine
;  9054          9152  1 !       that gets called after all the subscripts have been picked
;  9055          9153  1 !       up, to fix up these bounds.
;  9056          9154  1 !
;  9057          9155  1 ! INPUTS
;  9058          9156  1 !       PRIMPTR - A pointer to the Primary Descriptor being constructed.
;  9059          9157  1 !
;  9060          9158  1 ! OUTPUTS
;  9061          9159  1 !       The Primary Descriptor pointed to by PRIMPTR may be modified.
;  9062          9160  1 !
;  9063          9161  2     BEGIN
;  9064          9162  2     MAP
;  9065          9163  2         PRIMPTR: REF DBG$PRIMARY;
;  9066          9164  2
;  9067          9165  2     LOCAL
;  9068          9166  2         NODEPTR: REF DBG$PRIM_NODE,
;  9069          9167  2         SUBVECTOR: REF DBG$PRIM_NODE_SUBS;
;  9070          9168  2
;  9071          9169  2
;  9072          9170  2     ! Obtain a pointers to the Primary Descriptor Subnode and the
;  9073          9171  2     ! subscript vector within that subnode. Do nothing if the
;  9074          9172  2     ! subnode is not for an array.
;  9075          9173  2     !
;  9076          9174  2     NODEPTR = .PRIMPTR[DBG$L_PRIM_BLINK];
;  9077          9175  2     IF .NODEPTR[DBG$B_PNODE_FCODE] NEQ RST$K_TYPE_ARRAY
;  9078          9176  2     THEN
;  9079          9177  2         RETURN;
;  9080          9178  2     SUBVECTOR = NODEPTR[DBG$A_PNARR_SVECTOR];
;  9081          9179  2
;  9082          9180  2
;  9083          9181  2     ! Check for the subscript count being less than the dimension count,
;  9084          9182  2     ! but the range bit not being set. The point here is that we want
;  9085          9183  2     ! to treat this case as if it really were a range (e.g., treat
;  9086          9184  2     ! X[1] as being the same as X[1][1:10]. Note that this same code
;  9087          9185  2     ! appears in GET_SUBSCRIPTS when we discover that we do have a range.
;  9088          9186  2     !
;  9089          9187  2     IF  .NODEPTR[DBG$B_PNARR_SUBCNT] LSS .NODEPTR[DBG$B_PNARR_DIMCNT]
;  9090          9188  2     AND NOT .NODEPTR[DBG$V_PNARR_RANGE]
;  9091          9189  2     THEN
;  9092          9190  3         BEGIN
;  9093          9191  3         NODEPTR[DBG$V_PNARR_RANGE] = TRUE;
;  9094          9192  3         INCR I FROM 0 TO .NODEPTR[DBG$B_PNARR_SUBCNT] - 1 DO
;  9095          9193  4             BEGIN
;  9096          9194  4             SUBVECTOR[.I, DBG$L_PNSUB_LBOUND] =
;  9097          9195  4                 .SUBVECTOR[.I, DBG$L_PNSUB_SVALUE];
;  9098          9196  4             SUBVECTOR[.I, DBG$L_PNSUB_UBOUND] =
;  9099          9197  4                 .SUBVECTOR[.I, DBG$L_PNSUB_SVALUE];
```

```
;  9100          9198  3            END;
;  9101          9199  2        END;
;  9102          9200  2    RETURN;
;  9103          9201  1    END;


                                        000C 00000 FIX_UP_PRIMARY:
                                                                .WORD   Save R2,R3                              ;  9141
                                50      04    AC  D0 00002              MOVL    PRIMPTR, R0                   ;  9174
                                51      18    A0  D0 00006              MOVL    24(R0), NODEPTR               ;
                                01      09    A1  91 0000A              CMPB    9(NODEPTR), #1                ;  9175
                                        39    12 0000E                  BNEQ    3$                            ;
                                50      28    A1  9E 00010              MOVAB   40(R1), SUBVECTOR             ;  9178
                            1B  A1      1F    A1  91 00014              CMPB    31(NODEPTR), 27(NODEPTR)      ;  9187
                                        2E    1E 00019                  BGEQU   3$                            ;
                        29      0A  A1          03  E0 0001B             BBS     #3, 10(NODEPTR), 3$           ;  9188
                                0A  A1          08  88 00020             BISB2   #8, 10(NODEPTR)               ;  9191
                                53      1F    A1  9A 00024              MOVZBL  31(NODEPTR), R3               ;  9192
                                51            01  CE 00028              MNEGL   #1, I                         ;  9194
                                              18  11 0002B              BRB     2$                            ;
                        52              51    14  C5 0002D 1$:          MULL3   #20, I, R2                    ;
                                    08  A240  9F 00031                  PUSHAB  8(R2)[SUBVECTOR]              ;  9195
                                        6240  9F 00035                  PUSHAB  (R2)[SUBVECTOR]               ;
                                9E            9E  D0 00038              MOVL    @(SP)+, @(SP)+                ;
                                    0C  A240  9F 0003B                  PUSHAB  12(R2)[SUBVECTOR]             ;  9197
                                        6240  9F 0003F                  PUSHAB  (R2)[SUBVECTOR]               ;
                                9E            9E  D0 00042              MOVL    @(SP)+, @(SP)+                ;
                        E4              51    53  F2 00045 2$:          AOBLSS  R3, I, 1$                     ;  9192
                                              04 00049 3$:              RET                                   ;  9201

; Routine Size:  74 bytes,    Routine Base:  DBG$CODE + 26F4
```

```
;  9105          9202  1 ROUTINE GET_BLISS_SUBSCRIPTS(PRIMPTR, NAME): NOVALUE =
;  9106          9203  1 !
;  9107          9204  1 ! FUNCTION
;  9108          9205  1 !       This routine picks up subscript values in a BLISS structure
;  9109          9206  1 !       reference (i.e., any BLISS primary of the form X[...]).
;  9110          9207  1 !       It calls DBG$EXPRESSION_PARSER to pick up each subscript.
;  9111          9208  1 !
;  9112          9209  1 !       The kinds of BLISS structures we accept are:
;  9113          9210  1 !
;  9114          9211  1 !               X[i]            bitvector
;  9115          9212  1 !               X[i]            vector
;  9116          9213  1 !               X[n,p,s,e]      block
;  9117          9214  1 !               X[m,n,p,s,e]    blockvector
;  9118          9215  1 !
;  9119          9216  1 !       This routine assumes that the opening subscript bracket has
;  9120          9217  1 !       already been found and that the parse pointer points to the start
;  9121          9218  1 !       of the first subscript expression.  When this routine returns,
;  9122          9219  1 !       the parse pointer is left pointing at the first character after
;  9123          9220  1 !       the closing subscript bracket.
;  9124          9221  1 !
;  9125          9222  1 ! INPUTS
;  9126          9223  1 !       PRIMPTR - A pointer to the Primary Descriptor for a structure about
;  9127          9224  1 !                 to be subscripted.
;  9128          9225  1 !       NAME    - The name of the object being subscripted (for error
;  9129          9226  1 !                 message purposes)
;  9130          9227  1 !
;  9131          9228  1 ! OUTPUTS
;  9132          9229  1 !       The PRIMPTR Primary Descriptor is changed to include the subscript
;  9133          9230  1 !                 information. This is represented as follows:
;  9134          9231  1 !
;  9135          9232  1 !       X[i] bitvector          - i is stored in a Primary Descriptor Array
;  9136          9233  1 !                                 sub-node.
;  9137          9234  1 !       X[i] vector             - i is stored in a Primary Descriptor Array
;  9138          9235  1 !                                 sub-node.
;  9139          9236  1 !       X[n,p,s,e] block        - n is stored in a Primary Descriptor Array
;  9140          9237  1 !                                 sub-node. p, s, and e are stored in the
;  9141          9238  1 !                                 Primary Descriptor Root node, in the
;  9142          9239  1 !                                 prim_offset, prim_length, and sgnext fields.
;  9143          9240  1 !       X[m,n,p,s,e] blockvector- m and n are stored in a Primary Descriptor
;  9144          9241  1 !                                 Array sub-node. p, s, and e are stored in the
;  9145          9242  1 !                                 Primary Descriptor Root node, in the
;  9146          9243  1 !                                 prim_offset, prim_length, and sgnext fields.
;  9147          9244  1 !
;  9148          9245  2     BEGIN
;  9149          9246  2
;  9150          9247  2     MAP
;  9151          9248  2         PRIMPTR: REF DBG$PRIMARY;       ! Pointer to BLISS structure Primary
;  9152          9249  2                                         !    Descriptor.
;  9153          9250  2
;  9154          9251  2     LOCAL
;  9155          9252  2         COUNT,                          ! Count of field values
;  9156          9253  2         DECLTYPE: REF DBG$VALDESC,      ! Pointer to value descriptor for the
;  9157          9254  2                                         !    subscript value.
;  9158          9255  2         DSTPTR: REF DST$RECORD,         ! Pointer to BLISS structure DST entry
;  9159          9256  2         FCODE,                          ! Data type FCODE for current symbol
;  9160          9257  2         LA_PTR: REF VECTOR[,BYTE],      ! Lookahead pointer into input
;  9161          9258  2         LOW_RANGE_VAL,                  ! The lower value of a subscript range
```

```
;9162     9259  2          NODEPTR: REF DBG$PRIM_NODE,      ! Pointer to Prim Desc Array sub-node
;9163     9260  2          NODESUBPTR: REF DBG$PRIM_NODE_SUBS, ! Pointer to subscript blockvector
;9164     9261  2                                           !       in Prim Desc Array sub-node
;9165     9262  2          PTR: REF VECTOR[,LONG],          ! Temporary pointer to field name values
;9166     9263  2          REF_FLAG,                        ! True for REF objects
;9167     9264  2          RSTPTR: REF RST$ENTRY,           ! Pointer to BLISS structure RST entry
;9168     9265  2          SAVED_RADIX,                     ! Temporarily saved expression radix
;9169     9266  2          STRIDE,                          ! Stride for blocks and blockvectors
;9170     9267  2          STRUC,                           ! Code for kind of BLISS structure
;9171     9268  2          SUBSCR_COUNT,                    ! Count of the number of subscripts
;9172     9269  2          SUBVECTOR: VECTOR[5],            ! Vector of subscripts
;9173     9270  2          TOKEN,                           ! Pointer to a Lexical Token
;9174     9271  2          TYPEID,                          ! Pointer to a typeid
;9175     9272  2          VALPTR: REF DBG$VALDESC,         ! Pointer to subscript Value Descriptor
;9176     9273  2          VALUE;                           ! Subscript value
;9177     9274  2
;9178     9275  2      DBG$GL_CURRENT_PRIMARY = .PRIMPTR;
;9179     9276  2
;9180     9277  2      ! Check that the Primary Descriptor has the correct Kind and FCODE.
;9181     9278  2      ! If so, obtain a pointer to the DST record for this BLISS structure.
;9182     9279  2      !
;9183     9280  2      RSTPTR = .PRIMPTR [DBG$L_DHDR_SYMIDO];
;9184     9281  2      IF .RSTPTR EQL 0
;9185     9282  2      THEN
;9186     9283  2          SIGNAL(DBG$_NOTASTRUCT, 1, .NAME);
;9187     9284  2      DSTPTR = .RSTPTR [RST$L_DSTPTR];
;9188     9285  2      IF .PRIMPTR [DBG$B_DHDR_KIND] NEQ RST$K_DATA
;9189     9286  2      THEN
;9190     9287  2          SIGNAL (DBG$_NOTASTRUCT, 1, DSTPTR[DST$B_NAME]);
;9191     9288  2
;9192     9289  2      DBG$STA_SYMTYPE (.RSTPTR, FCODE, TYPEID);
;9193     9290  2      IF .FCODE NEQ RST$K_TYPE_BLIDATA
;9194     9291  2      THEN
;9195     9292  2          SIGNAL (DBG$_NOTASTRUCT, 1, DSTPTR[DST$B_NAME]);
;9196     9293  2
;9197     9294  2
;9198     9295  2      ! Check for no field structure in the RST - this arises when the
;9199     9296  2      ! user defines his own BLISS structure, instead of using one of
;9200     9297  2      ! the builtins BITVECTOR, VECTOR, BLOCK, or BLOCKVECTOR. We do
;9201     9298  2      ! not handle this case.
;9202     9299  2      !
;9203     9300  2      STRUC = .DSTPTR [DST$V_BLI_STRUC];
;9204     9301  2      IF .STRUC EQL DST$K_BLI_NOSTRUC
;9205     9302  2      THEN
;9206     9303  2          SIGNAL(DBG$_NOTASTRUCT, 1, DSTPTR[DST$B_NAME]);
;9207     9304  2
;9208     9305  2
;9209     9306  2      ! If the REF bit is set, then there is a sub-node for the dereferencing.
;9210     9307  2      ! Call DBG$BUILD_PRIMARY_SUBNODE to build a new subnode for the array
;9211     9308  2      ! information. Light the EVAL bit in the dereference subnode.
;9212     9309  2      !
;9213     9310  2      IF .DSTPTR [DST$V_BLI_REF]
;9214     9311  2      THEN
;9215     9312  3          BEGIN
;9216     9313  3          DBG$BUILD_PRIMARY_SUBNODE (.PRIMPTR, RST$K_DATA, 0, RST$K_TYPE_BLIDATA,
;9217     9314  3                                     .PRIMPTR [DBG$L_DHDR_TYPEID], .DSTPTR);
;9218     9315  3          NODEPTR = .PRIMPTR [DBG$L_PRIM_FLINK];
```

```
;  9219          9316  3          NODEPTR [DBG$V_PNODE_EVAL] = TRUE;
;  9220          9317  3          REF_FLAG = TRUE;
;  9221          9318  3          END
;  9222          9319  2      ELSE
;  9223          9320  2          REF_FLAG = FALSE;
;  9224          9321  2
;  9225          9322  2
;  9226          9323  2      ! Obtain pointers to the Primary Descriptor Sub-Node and the subscript
;  9227          9324  2      ! blockvector within that node.  Light the eval bit in the subnode.
;  9228          9325  2      !
;  9229          9326  2      NODEPTR = .PRIMPTR [DBG$L_PRIM_BLINK];
;  9230          9327  2      NODESUBPTR = NODEPTR [DBG$A_PNARR_SVECTOR];
;  9231          9328  2      NODEPTR [DBG$V_PNODE_EVAL] = TRUE;
;  9232          9329  2
;  9233          9330  2
;  9234          9331  2      ! Loop through the list of subscripts.  Pick up each actual subscript and
;  9235          9332  2      ! add it to the subscript vector.  When we get to the closing subscript
;  9236          9333  2      ! parenthesis, we check all the subscript values for validity and complete
;  9237          9334  2      ! the Primary Descriptor Sub-Node accordingly.
;  9238          9335  2      !
;  9239          9336  2      SUBSCR_COUNT = 0;
;  9240          9337  2      TERMINATOR_CODE = TOKEN$K_TERM_COMMA;
;  9241          9338  2      WHILE .TERMINATOR_CODE NEQ TOKEN$K_TERM_CLOSE DO
;  9242          9339  3          BEGIN
;  9243          9340  3
;  9244          9341  3
;  9245          9342  3          ! Look for the asterisk. X[*] is the same as X[lower:upper].
;  9246          9343  3          ! If we find the asterisk then advance the character pointer beyond
;  9247          9344  3          ! the asterisk and also increment the subscript count.
;  9248          9345  3          !
;  9249          9346  3          LA_PTR = .CHARPTR;
;  9250          9347  3          WHILE .LA_PTR[0] EQL ' ' DO LA_PTR = .LA_PTR + 1;
;  9251          9348  3          IF .LA_PTR[0] EQL '*'
;  9252          9349  3          THEN
;  9253          9350  4              BEGIN
;  9254          9351  4              IF .NODEPTR[DBG$V_PNARR_RANGE] OR
;  9255          9352  4                (.SUBSCR_COUNT NEQ 0)        OR
;  9256          9353  5               ((.STRUC NEQ DST$K_BLI_BITVEC) AND
;  9257          9354  5                (.STRUC NEQ DST$K_BLI_VEC)    AND
;  9258          9355  5                (.STRUC NEQ DST$K_BLI_BLKVEC))
;  9259          9356  4              THEN
;  9260          9357  4                  SIGNAL(DBG$_INVRANSPEC);
;  9261          9358  4
;  9262          9359  4              CHARPTR = .LA_PTR + 1;
;  9263          9360  4              SUBSCR_COUNT = .SUBSCR_COUNT + 1;
;  9264          9361  4
;  9265          9362  4              ! Call the Lexical Scanner to take us past the ',' or
;  9266          9363  4              ! or ']' or ')'. This will set TERMINATOR_CODE to the
;  9267          9364  4              ! terminator that is seen. If we do not see a terminator
;  9268          9365  4              ! then signal a syntax error. Also signal an error if
;  9269          9366  4              ! ':' was the terminator.
;  9270          9367  4              !
;  9271          9368  4              TOKEN = DBG$LEXICAL_SCANNER (FALSE, FALSE,
;  9272          9369  4                          .SUBSCRIPT_TERM_TBL, 0);
;  9273          9370  4              IF .TOKEN NEQ TERMINATOR_TOKEN
;  9274          9371  4              THEN
;  9275          9372  5                  BEGIN
```

```
;  9276     9373  5                LOCAL
;  9277     9374  5                    ASCIC_STRING: VECTOR[2,BYTE];
;  9278     9375  5                ASCIC_STRING[0] = 1;
;  9279     9376  5                ASCIC_STRING[1] = .CHARPTR[0];
;  9280     9377  5                SIGNAL(DBG$_SYNERREXPR, 1, ASCIC_STRING);
;  9281     9378  4                END;
;  9282     9379  4            IF .TERMINATOR_CODE EQL TOKEN$K_TERM_COLON
;  9283     9380  4            THEN
;  9284     9381  4                SIGNAL (DBG$_INVRANSPEC);
;  9285     9382  4            IF .TERMINATOR_CODE EQL TOKEN$K_TERM_NONE
;  9286     9383  4            THEN
;  9287     9384  4                SIGNAL(DBG$_MISCLOSUB);
;  9288     9385  4            CHARPTR = .CHARPTR + .TERMINATOR_LENGTH;
;  9289     9386  4
;  9290     9387  4
;  9291     9388  4            ! Turn this reference into a range.
;  9292     9389  4            !
;  9293     9390  4            NODEPTR[DBG$V_PNARR_RANGE] = TRUE;
;  9294     9391  4            LOW_RANGE_VAL = .NODESUBPTR[0, DBG$L_PNSUB_LBOUND];
;  9295     9392  4            SUBVECTOR[0] = .NODESUBPTR[0, DBG$L_PNSUB_UBOUND];
;  9296     9393  4            END
;  9297     9394  4
;  9298     9395  3        ELSE
;  9299     9396  4            BEGIN
;  9300     9397  4
;  9301     9398  4
;  9302     9399  4            ! Call the Expression Parser to pick up the next subscript.  Note that
;  9303     9400  4            ! we set the radix to decimal over this call and then restore it.  Also
;  9304     9401  4            ! note that OWN variables TERMINATOR_CODE and TERMINATOR_LENGTH are set
;  9305     9402  4            ! by the Expression Parser as side-effects.
;  9306     9403  4            !
;  9307     9404  4            SAVED_RADIX = .EXPRESSION_RADIX;
;  9308     9405  4            EXPRESSION_RADIX = DBG$K_DECIMAL;
;  9309     9406  4            VALPTR = DBG$EXPRESSION_PARSER (FALSE, .SUBSCRIPT_TERM_TBL);
;  9310     9407  4            EXPRESSION_RADIX = .SAVED_RADIX;
;  9311     9408  4
;  9312     9409  4
;  9313     9410  4            ! Check the terminator code.  If there was no terminator (i.e., the
;  9314     9411  4            ! input line just ended), signal an error.  Otherwise we got a comma,
;  9315     9412  4            ! colon, or closing bracket and we increment CHARPTR to get past it.
;  9316     9413  4            !
;  9317     9414  4            IF .TERMINATOR_CODE EQL TOKEN$K_TERM_NONE THEN SIGNAL(DBG$_MISCLOSUB);
;  9318     9415  4            CHARPTR = .CHARPTR + .TERMINATOR_LENGTH;
;  9319     9416  4
;  9320     9417  4
;  9321     9418  4            ! Check whether this "subscript" is actually a BLISS field name.  If
;  9322     9419  4            ! so, set up a pointer to the BLISS field components and copy those
;  9323     9420  4            ! four components into the subscript vector.
;  9324     9421  4            !
;  9325     9422  4            IF .VALPTR [DBG$B_DHDR_FCODE] EQL RST$K_TYPE_BLIFLD
;  9326     9423  4            THEN
;  9327     9424  5                BEGIN
;  9328     9425  6                IF .NODEPTR[DBG$V_PNARR_RANGE] AND (.SUBSCR_COUNT EQL 0)
;  9329     9426  5                THEN
;  9330     9427  5                    SIGNAL(DBG$_INVRANSPEC);
;  9331     9428  5
;  9332     9429  5                PTR = VALPTR[DBG$A_VALUE_ADDRESS];
```

```
;  9333          9430  5                 COUNT = .PTR[0];
;  9334          9431  5                 INCR I FROM 1 TO .COUNT DO
;  9335          9432  6                     BEGIN
;  9336          9433  6                     IF .SUBSCR_COUNT LSS 5 THEN SUBVECTOR[.SUBSCR_COUNT] = .PTR[.I];
;  9337          9434  6                     SUBSCR_COUNT = .SUBSCR_COUNT + 1;
;  9338          9435  5                     END;
;  9339          9436  5
;  9340          9437  5                 END
;  9341          9438  5
;  9342          9439  5
;  9343          9440  5             ! The subscript is not a BLISS field name so we pick up the subscript
;  9344          9441  5             ! value and convert it to integer.  This subscript value is then put
;  9345          9442  5             ! away in the subscript vector and the subscript count is incremented.
;  9346          9443  5             !
;  9347          9444  4             ELSE
;  9348          9445  5                 BEGIN
;  9349          9446  5
;  9350          9447  5
;  9351          9448  5                 ! Convert the value descriptor to an integer value.
;  9352          9449  5                 !
;  9353          9450  5                 VALUE = CONVERT_TO_INTEGER (.VALPTR);
;  9354          9451  5
;  9355          9452  5
;  9356          9453  5                 ! If the terminator is a colon, we have the first part of a sub-
;  9357          9454  5                 ! script range specification (such as "ARR[2:5]").  See if a range
;  9358          9455  5                 ! specification is allowed (it is only allowed on the first sub-
;  9359          9456  5                 ! script of a bitvector, a normal vector, or a blockvector) and
;  9360          9457  5                 ! save away this lower value of the range.
;  9361          9458  5                 !
;  9362          9459  5                 IF .TERMINATOR_CODE EQL TOKEN$K_TERM_COLON
;  9363          9460  5                 THEN
;  9364          9461  6                     BEGIN
;  9365          9462  6                     IF .NODEPTR[DBG$V_PNARR_RANGE] OR
;  9366          9463  6                       (.SUBSCR_COUNT NEQ 0)       OR
;  9367          9464  7                      ((.STRUC NEQ DST$K_BLI_BITVEC) AND
;  9368          9465  7                       (.STRUC NEQ DST$K_BLI_VEC)    AND
;  9369          9466  7                       (.STRUC NEQ DST$K_BLI_BLKVEC))
;  9370          9467  6                     THEN
;  9371          9468  6                         SIGNAL(DBG$_INVRANSPEC);
;  9372          9469  6
;  9373          9470  6                     NODEPTR[DBG$V_PNARR_RANGE] = TRUE;
;  9374          9471  6                     LOW_RANGE_VAL = .VALUE;
;  9375          9472  6                     END
;  9376          9473  6
;  9377          9474  6
;  9378          9475  6                 ! The terminator is not a colon, so we have the complete specifica-
;  9379          9476  6                 ! tion for the current subscript.  Here we simply fill the subscript
;  9380          9477  6                 ! value (or the upper bound of the range) into SUBVECTOR and incre-
;  9381          9478  6                 ! ment the subscript count.  We can ignore subscripts after the
;  9382          9479  6                 ! fifth one because they will be in error anyway.
;  9383          9480  6                 !
;  9384          9481  5                 ELSE
;  9385          9482  6                     BEGIN
;  9386          9483  6                     IF .SUBSCR_COUNT LSS 5
;  9387          9484  6                     THEN
;  9388          9485  6                         SUBVECTOR[.SUBSCR_COUNT] = .VALUE;
;  9389          9486  6
```

```
;  9390     9487  6                    SUBSCR_COUNT = .SUBSCR_COUNT + 1;
;  9391     9488  5                    END;
;  9392     9489  4                END;
;  9393     9490  3            END;
;  9394     9491  3
;  9395     9492  2        END;                            ! End of WHILE loop over subscripts
;  9396     9493  2
;  9397     9494  2
;  9398     9495  2        ! We found the end of the subscript list, i.e. the closing subscript
;  9399     9496  2        ! parenthesis.  Now case on the kind of BLISS structure we are dealing
;  9400     9497  2        ! with and check the subscript values and build the Primary Descriptor
;  9401     9498  2        ! Sub-Node accordingly.
;  9402     9499  2        !
;  9403     9500  2        CASE .STRUC FROM DST$K_BLI_NOSTRUC TO DST$K_BLI_BLKVEC OF
;  9404     9501  2            SET
;  9405     9502  2
;  9406     9503  2
;  9407     9504  2            ! Handle the Bitvector case.
;  9408     9505  2            !
;  9409     9506  2            [DST$K_BLI_BITVEC]:
;  9410     9507  3                BEGIN
;  9411     9508  3
;  9412     9509  3
;  9413     9510  3                ! Check that the bit-vector had exactly one subscript.
;  9414     9511  3                !
;  9415     9512  3                IF .SUBSCR_COUNT LSS 1 THEN SIGNAL(DBG$_TOOFEWSUB, 1, 1);
;  9416     9513  3                IF .SUBSCR_COUNT GTR 1 THEN SIGNAL(DBG$_TOOMANSUB, 1, 1);
;  9417     9514  3
;  9418     9515  3
;  9419     9516  3                ! Check that the subscript value is in range.
;  9420     9517  3                !
;  9421     9518  3                IF NOT .REF_FLAG
;  9422     9519  3                THEN
;  9423     9520  4                    BEGIN
;  9424     9521  4                    IF (.SUBVECTOR[0] LSS 0) OR
;  9425     9522  5                       (.SUBVECTOR[0] GEQ .DSTPTR[DST$L_BLI_BITVEC_SIZE])
;  9426     9523  4                    THEN
;  9427     9524  4                        SIGNAL (DBG$_STRUCSIZE, 2,
;  9428     9525  4                                .DSTPTR[DST$L_BLI_BITVEC_SIZE], .SUBVECTOR[0]);
;  9429     9526  4
;  9430     9527  4
;  9431     9528  4                    ! If a subscript range was specified, check that the lower range
;  9432     9529  4                    ! value is also in range.
;  9433     9530  4                    !
;  9434     9531  4                    IF .NODEPTR[DBG$V_PNARR_RANGE] AND
;  9435     9532  5                     ((.LOW_RANGE_VAL LSS 0) OR
;  9436     9533  5                      (.LOW_RANGE_VAL GEQ .DSTPTR[DST$L_BLI_BITVEC_SIZE]))
;  9437     9534  4                    THEN
;  9438     9535  4                        SIGNAL (DBG$_STRUCSIZE, 2,
;  9439     9536  4                                .DSTPTR[DST$L_BLI_BITVEC_SIZE], .LOW_RANGE_VAL);
;  9440     9537  3                    END;
;  9441     9538  3
;  9442     9539  3
;  9443     9540  3                ! Fill in the Primary Descriptor Array Sub-Node.
;  9444     9541  3                !
;  9445     9542  3                NODEPTR [DBG$B_PNARR_SUBCNT] = 1;
;  9446     9543  3                NODESUBPTR [0, DBG$L_PNSUB_SVALUE] = .SUBVECTOR[0];
```

```
; 9447          9544  3
; 9448          9545  2          END;                            ! End of bitvector case
; 9449          9546  2
; 9450          9547  2
; 9451          9548  2      ! Handle ordinary BLISS vectors.
; 9452          9549  2      !
; 9453          9550  2      [DST$K_BLI_VEC]:
; 9454          9551  2          BEGIN
; 9455          9552  2
; 9456          9553  2
; 9457          9554  2          ! Check that the vector had exactly one subscript.
; 9458          9555  2          !
; 9459          9556  2          IF .SUBSCR_COUNT LSS 1 THEN SIGNAL(DBG$_TOOFEWSUB, 1, 1);
; 9460          9557  2          IF .SUBSCR_COUNT GTR 1 THEN SIGNAL(DBG$_TOOMANSUB, 1, 1);
; 9461          9558  2
; 9462          9559  2
; 9463          9560  2          ! Check that the subscript value is in range.
; 9464          9561  2          !
; 9465          9562  2          IF NOT .REF_FLAG
; 9466          9563  3          THEN
; 9467          9564  4              BEGIN
; 9468          9565  4              IF (.SUBVECTOR[0] LSS 0) OR
; 9469          9566  5                 (.SUBVECTOR[0] GEQ .DSTPTR[DST$L_BLI_VEC_UNITS])
; 9470          9567  4              THEN
; 9471          9568  4                  SIGNAL(DBG$_STRUCSIZE, 2,
; 9472          9569  4                         .DSTPTR[DST$L_BLI_VEC_UNITS], .SUBVECTOR[0]);
; 9473           570  4
; 9474          9571  4
; 9475          9572  4                  ! If a subscript range was specified, check that the lower range
; 9476          9573  4                  ! value is also in range.
; 9477          9574  4                  !
; 9478          9575  4                  IF .NODEPTR[DBG$V_PNARR_RANGE] AND
; 9479          9576  5                   ((.LOW_RANGE_VAL LSS 0) OR
; 9480          9577  5                    (.LOW_RANGE_VAL GEQ .DSTPTR[DST$L_BLI_VEC_UNITS]))
; 9481          9578  4                  THEN
; 9482          9579  4                      SIGNAL(DBG$_STRUCSIZE, 2,
; 9483          9580  4                             .DSTPTR[DST$L_BLI_VEC_UNITS], .LOW_RANGE_VAL);
; 9484          9581  3                  END;
; 9485          9582  3
; 9486          9583  3
; 9487          9584  3              ! Fill in the Primary Descriptor Sub-Node.
; 9488          9585  3              !
; 9489          9586  3              NODEPTR [DBG$B_PNARR_SUBCNT] = 1;
; 9490          9587  3              NODESUBPTR [0, DBG$L_PNSUB_SVALUE] = .SUBVECTOR[0];
; 9491          9588  3
; 9492          9589  2              END;                        ! End of BLISS vector case
; 9493          9590  2
; 9494          9591  2
; 9495          9592  2      ! Handle BLISS blocks.
; 9496          9593  2      !
; 9497          9594  2      [DST$K_BLI_BLOCK]:
; 9498          9595  3          BEGIN
; 9499          9596  3
; 9500          9597  3          ! Fill in the correct information for the array subnode
; 9501          9598  3          ! (It was dummied up as an array of longwords for purposes
; 9502          9599  3          !  of aggregate output earlier).
; 9503          9600  3          !
```

```
;  9504          9601  3          STRIDE = .DSTPTR [DST$V_BLI_BLOCK_UNIT_SIZE];
;  9505          9602  3          NODESUBPTR [0, DBG$L_PNSUB_STRIDE] = .STRIDE;
;  9506          9603  3          NODESUBPTR [0, DBG$L_PNSUB_UBOUND] =
;  9507          9604  3              .DSTPTR [DST$L_BLI_BLOCK_UNITS]-1;
;  9508          9605  3          NODEPTR[DBG$W_PNARR_LENGTH] = .STRIDE;
;  9509          9606  3          IF .STRIDE EQL 1
;  9510          9607  3          THEN
;  9511          9608  3              IF .SUBVECTOR[3]
;  9512          9609  3              THEN
;  9513          9610  3                  NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_B
;  9514          9611  3              ELSE
;  9515          9612  3                  NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_BU
;  9516          9613  3          ELSE IF .STRIDE EQL 2
;  9517          9614  3          THEN
;  9518          9615  3              IF .SUBVECTOR[3]
;  9519          9616  3              THEN
;  9520          9617  3                  NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_W
;  9521          9618  3              ELSE
;  9522          9619  3                  NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_WU
;  9523          9620  3          ELSE IF .STRIDE EQL 4
;  9524          9621  3          THEN
;  9525          9622  3              IF .SUBVECTOR[3]
;  9526          9623  3              THEN
;  9527          9624  3                  NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_L
;  9528          9625  3              ELSE
;  9529          9626  3                  NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_LU
;  9530          9627  3          ELSE
;  9531          9628  4              BEGIN
;  9532          9629  4              IF .SUBVECTOR[3]
;  9533          9630  4              THEN
;  9534          9631  4                  NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_V
;  9535          9632  4              ELSE
;  9536          9633  4                  NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_VU;
;  9537          9634  4              NODEPTR[DBG$W_PNARR_LENGTH] = 8 * .NODEPTR[DBG$W_PNARR_LENGTH];
;  9538          9635  3              END;
;  9539          9636  3
;  9540          9637  3          ! Check that the block had exactly four subscripts.
;  9541          9638  3          !
;  9542          9639  3          IF .SUBSCR_COUNT LSS 4 THEN SIGNAL(DBG$_TOOFEWSUB, 1, 4);
;  9543          9640  3          IF .SUBSCR_COUNT GTR 4 THEN SIGNAL(DBG$_TOOMANSUB, 1, 4);
;  9544          9641  3
;  9545          9642  3
;  9546          9643  3          ! Check that the subscript values are in range.
;  9547          9644  3          !
;  9548          9645  3          IF NOT .REF_FLAG
;  9549          9646  3          THEN
;  9550          9647  3              IF (.SUBVECTOR[0] LSS 0) OR
;  9551          9648  4                 (.SUBVECTOR[0] GEQ .DSTPTR[DST$L_BLI_BLOCK_UNITS])
;  9552          9649  3              THEN
;  9553          9650  3                  SIGNAL(DBG$_STRUCSIZE, 2,
;  9554          9651  3                         .DSTPTR[DST$L_BLI_BLOCK_UNITS], .SUBVECTOR[0]);
;  9555          9652  3
;  9556          9653  3          IF (.SUBVECTOR[1] LSS -%X'8000') OR
;  9557          9654  4             (.SUBVECTOR[1] GTR %X'7FFF')
;  9558          9655  3          THEN
;  9559          9656  3              SIGNAL(DBG$_ILLOFFSET, 1, .SUBVECTOR[1]);
;  9560          9657  3
```

```
;  9561          9658  4              IF (.SUBVECTOR[2] LSS 0)
;  9562          9659  3              THEN
;  9563          9660  3                  SIGNAL(DBG$_ILLLENGTH, 1, .SUBVECTOR[2]);
;  9564          9661  3
;  9565          9662  4              IF (.SUBVECTOR[2] GTR 32)
;  9566          9663  3              THEN
;  9567          9664  4                  BEGIN
;  9568          9665  4                  SUBVECTOR[2] = 32;
;  9569          9666  4                  SIGNAL(DBG$_SIZETRUNC);
;  9570          9667  3                  END;
;  9571          9668  3
;  9572          9669  4              IF (.SUBVECTOR[3] NEQ 0) AND (.SUBVECTOR[3] NEQ 1)
;  9573          9670  3              THEN
;  9574          9671  3                  SIGNAL(DBG$_ILLSIGEXT, 1, .SUBVECTOR[3]);
;  9575          9672  3
;  9576          9673  3
;  9577          9674  3              ! Fill in the Primary Descriptor Sub-Node.
;  9578          9675  3              !
;  9579          9676  3              PRIMPTR [DBG$V_DHDR_BLIBLK] = TRUE;
;  9580          9677  3              PRIMPTR [DBG$V_DHDR_SUBREF] = TRUE;
;  9581          9678  3              PRIMPTR [DBG$V_DHDR_BITREF] = TRUE;
;  9582          9679  3              PRIMPTR [DBG$W_PRIM_OFFSET] = .SUBVECTOR[1];
;  9583          9680  3              PRIMPTR [DBG$W_PRIM_LENGTH] = .SUBVECTOR[2];
;  9584          9681  3              PRIMPTR [DBG$V_DHDR_SGNEXT] = .SUBVECTOR[3];
;  9585          9682  3              NODEPTR [DBG$B_PNARR_SUBCNT] = 1;
;  9586          9683  3              NODESUBPTR [0, DBG$L_PNSUB_SVALUE] = .SUBVECTOR[0];
;  9587          9684  3
;  9588          9685  2              END;                            ! End of BLISS block case
;  9589          9686  2
;  9590          9687  2
;  9591          9688  2          ! Handle the Blockvector case.
;  9592          9689  2          !
;  9593          9690  2          [DST$K_BLI_BLKVEC]:
;  9594          9691  3              BEGIN
;  9595          9692  3
;  9596          9693  3              ! We previously represented the block as a block of longwords,
;  9597          9694  3              ! for purposes of aggregate output. If we get here, however,
;  9598          9695  3              ! we are no longer doing aggregate output.
;  9599          9696  3              ! So, fix up the information
;  9600          9697  3              ! here, filling in the correct stride.
;  9601          9698  3              !
;  9602          9699  3              STRIDE = .DSTPTR [DST$B_BLI_BLKVEC_UNIT_SIZE];
;  9603          9700  3              NODESUBPTR [1, DBG$L_PNSUB_STRIDE] = .STRIDE;
;  9604          9701  3              NODESUBPTR [1, DBG$L_PNSUB_UBOUND] =
;  9605          9702  3                  .DSTPTR [DST$L_BLI_BLKVEC_UNITS]-1;
;  9606          9703  3              NODEPTR[DBG$W_PNARR_LENGTH] = .STRIDE;
;  9607          9704  3              IF .STRIDE EQL 1
;  9608          9705  3              THEN
;  9609          9706  3                  IF .SUBVECTOR[4]
;  9610          9707  3                  THEN
;  9611          9708  3                      NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_B
;  9612          9709  3                  ELSE
;  9613          9710  3                      NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_BU
;  9614          9711  3              ELSE IF .STRIDE EQL 2
;  9615          9712  3              THEN
;  9616          9713  3                  IF .SUBVECTOR[4]
;  9617          9714  3                  THEN
```

```
; 9618     9715  3                NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_W
; 9619     9716  3            ELSE
; 9620     9717  3                NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_WU
; 9621     9718  3        ELSE IF .STRIDE EQL 4
; 9622     9719  3        THEN
; 9623     9720  3            IF .SUBVECTOR[4]
; 9624     9721  3            THEN
; 9625     9722  3                NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_L
; 9626     9723  3            ELSE
; 9627     9724  3                NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_LU
; 9628     9725  3        ELSE
; 9629     9726  4            BEGIN
; 9630     9727  4            IF .SUBVECTOR[4]
; 9631     9728  4            THEN
; 9632     9729  4                NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_V
; 9633     9730  4            ELSE
; 9634     9731  4                NODEPTR [DBG$B_PNARR_DTYPE] = DSC$K_DTYPE_VU;
; 9635     9732  4            NODEPTR[DBG$W_PNARR_LENGTH] = 8 * .NODEPTR[DBG$W_PNARR_LENGTH];
; 9636     9733  3            END;
; 9637     9734  3
; 9638     9735  3        ! Check that the block-vector had exactly five subscripts.
; 9639     9736  3        !
; 9640     9737  3        IF .SUBSCR_COUNT LSS 5 THEN SIGNAL(DBG$_TOOFEWSUB, 1, 5);
; 9641     9738  3        IF .SUBSCR_COUNT GTR 5 THEN SIGNAL(DBG$_TOOMANSUB, 1, 5);
; 9642     9739  3
; 9643     9740  3
; 9644     9741  3        ! Check that the subscript values are in range.
; 9645     9742  3        !
; 9646     9743  3        IF NOT .REF_FLAG
; 9647     9744  3        THEN
; 9648     9745  4            BEGIN
; 9649     9746  4            IF (.SUBVECTOR[0] LSS 0) OR
; 9650     9747  5               (.SUBVECTOR[0] GEQ .DSTPTR[DST$L_BLI_BLKVEC_BLOCKS])
; 9651     9748  4            THEN
; 9652     9749  4                SIGNAL(DBG$_STRUCSIZE, 2,
; 9653     9750  4                       .DSTPTR[DST$L_BLI_BLKVEC_BLOCKS], .SUBVECTOR[0]);
; 9654     9751  4
; 9655     9752  4            IF (.SUBVECTOR[1] LSS 0) OR
; 9656     9753  5               (.SUBVECTOR[1] GEQ .DSTPTR[DST$L_BLI_BLKVEC_UNITS])
; 9657     9754  4            THEN
; 9658     9755  4                SIGNAL(DBG$_STRUCSIZE, 2,
; 9659     9756  4                       .DSTPTR[DST$L_BLI_BLKVEC_UNITS], .SUBVECTOR[1]);
; 9660     9757  3            END;
; 9661     9758  3
; 9662     9759  3        IF (.SUBVECTOR[2] LSS -%X'8000') OR
; 9663     9760  4           (.SUBVECTOR[2] GTR %X'7FFF')
; 9664     9761  3        THEN
; 9665     9762  3            SIGNAL(DBG$_ILLOFFSET, 1, .SUBVECTOR[2]);
; 9666     9763  3
; 9667     9764  4        IF (.SUBVECTOR[3] LSS 0)
; 9668     9765  3        THEN
; 9669     9766  3            SIGNAL(DBG$_ILLLENGTH, 1, .SUBVECTOR[3]);
; 9670     9767  3
; 9671     9768  4        IF (.SUBVECTOR[3] GTR 32)
; 9672     9769  3        THEN
; 9673     9770  4            BEGIN
; 9674     9771  4            SUBVECTOR[3] = 32;
```

```
;  9675     9772  4              SIGNAL(DBG$_SIZETRUNC);
;  9676     9773  3              END;
;  9677     9774  3
;  9678     9775  4          IF (.SUBVECTOR[4] NEQ 0) AND (.SUBVECTOR[4] NEQ 1)
;  9679     9776  3          THEN
;  9680     9777  3              SIGNAL(DBG$_ILLSIGEXT, 1, .SUBVECTOR[4]);
;  9681     9778  3
;  9682     9779  3
;  9683     9780  3          ! If a subscript range was specified, check that the lower range
;  9684     9781  3          ! value is also in range.
;  9685     9782  3          !
;  9686     9783  3          IF NOT .REF_FLAG
;  9687     9784  3          THEN
;  9688     9785  3              IF .NODEPTR[DBG$V_PNARR_RANGE] AND
;  9689     9786  4               ((.LOW_RANGE_VAL LSS 0) OR
;  9690     9787  4                (.LOW_RANGE_VAL GEQ .DSTPTR[DST$L_BLI_BLKVEC_BLOCKS]))
;  9691     9788  3              THEN
;  9692     9789  3                  SIGNAL(DBG$_STRUCSIZE, 2,
;  9693     9790  3                         .DSTPTR[DST$L_BLI_BLKVEC_BLOCKS], .LOW_RANGE_VAL);
;  9694     9791  3
;  9695     9792  3
;  9696     9793  3          ! Fill in the Primary Descriptor Sub-Node.
;  9697     9794  3          !
;  9698     9795  3          PRIMPTR [DBG$V_DHDR_BLIBLK] = TRUE;
;  9699     9796  3          PRIMPTR [DBG$V_DHDR_SUBREF] = TRUE;
;  9700     9797  3          PRIMPTR [DBG$V_DHDR_BITREF] = TRUE;
;  9701     9798  3          PRIMPTR [DBG$W_PRIM_OFFSET] = .SUBVECTOR[2];
;  9702     9799  3          PRIMPTR [DBG$W_PRIM_LENGTH] = .SUBVECTOR[3];
;  9703     9800  3          PRIMPTR [DBG$V_DHDR_SGNEXT] = .SUBVECTOR[4];
;  9704     9801  3          NODEPTR [DBG$B_PNARR_SUBCNT] = 2;
;  9705     9802  3          NODESUBPTR [0, DBG$L_PNSUB_SVALUE] = .SUBVECTOR[0];
;  9706     9803  3          NODESUBPTR [1, DBG$L_PNSUB_SVALUE] = .SUBVECTOR[1];
;  9707     9804  3
;  9708     9805  3              ! If there was a range on the first subscript then make
;  9709     9806  3              ! the second one into a range too.
;  9710     9807  3              !
;  9711     9808  3              IF .NODEPTR[DBG$V_PNARR_RANGE]
;  9712     9809  3              THEN
;  9713     9810  4                  BEGIN
;  9714     9811  4                  NODESUBPTR [1, DBG$L_PNSUB_LBOUND] = .SUBVECTOR[1];
;  9715     9812  4                  NODESUBPTR [1, DBG$L_PNSUB_UBOUND] = .SUBVECTOR[1];
;  9716     9813  3                  END;
;  9717     9814  2              END;                            ! End of BLISS block-vector case
;  9718     9815  2
;  9719     9816  2
;  9720     9817  2          ! Any other case should never occur and constitutes an internal
;  9721     9818  2          ! error in the BLISS Debug Symbol Table (DST).
;  9722     9819  2          !
;  9723     9820  2          [INRANGE, OUTRANGE]:
;  9724     9821  2              SIGNAL(DBG$_INVDSTREC);
;  9725     9822  2
;  9726     9823  2          TES;                                ! End of CASE of BLISS structure type
;  9727     9824  2
;  9728     9825  2
;  9729     9826  2      ! If a subscript range (as in ARR[2:5]) was specified for the first sub-
;  9730     9827  2      ! script, modify the array's lower and upper bounds in the Primary Descrip-
;  9731     9828  2      ! tor to represent the array "slice" specified by that subscript range.
```

```
; 9732        9829  2     !
; 9733        9830  2     IF .NODEPTR[DBG$V_PNARR_RANGE]
; 9734        9831  2     THEN
; 9735        9832  3         BEGIN
; 9736        9833  3         IF .LOW_RANGE_VAL GTR .SUBVECTOR[0] THEN SIGNAL(DBG$_INVRANSPEC);
; 9737        9834  3         NODESUBPTR[0, DBG$L_PNSUB_SVALUE] = .LOW_RANGE_VAL;
; 9738        9835  3         NODESUBPTR[0, DBG$L_PNSUB_LBOUND] = .LOW_RANGE_VAL;
; 9739        9836  3         NODESUBPTR[0, DBG$L_PNSUB_UBOUND] = .SUBVECTOR[0];
; 9740        9837  3         RETURN;
; 9741        9838  2         END;
; 9742        9839  2
; 9743        9840  2
; 9744        9841  2     ! Build a new subnode. The typeid that we pass in to
; 9745        9842  2     ! BUILD_PRIMARY_SUBNODE describes the element referenced by the
; 9746        9843  2     ! subscript expression. This typeid is pulled from the CELLTYPE field.
; 9747        9844  2     !
; 9748        9845  2     DBG$BUILD_PRIMARY_SUBNODE (.PRIMPTR, RST$K_DATA, 0,
; 9749        9846  2         RST$K_TYPE_ATOMIC, .NODEPTR[DBG$L_PNARR_CELLTYPE], 0);
; 9750        9847  2     RETURN;
; 9751        9848  1     END;


                                    0FFC 00000 GET_BLISS_SUBSCRIPTS:
                                                       .WORD   Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11        ; 9202
                                    5E          3C  C2 00002   SUBL2   #60, SP                             ;
                                    56      04  AC  D0 00005   MOVL    PRIMPTR, R6                         ; 9275
                          00000000G 00          56  D0 00009   MOVL    R6, DBG$GL_CURRENT_PRIMARY          ;
                                    52      0C  A6  D0 00010   MOVL    12(R6), RSTPTR                      ; 9280
                                                12  12 00014   BNEQ    1$                                  ; 9281
                                            08  AC  DD 00016   PUSHL   NAME                               ; 9283
                                            01  DD 00019   PUSHL   #1                                   ;
                                    000281A8    8F  DD 0001B   PUSHL   #164264                             ;
                                        00000000G   00  03  FB 00021   CALLS   #3, LIB$SIGNAL             ;
                                                54      0C  A2  D0 00028 1$:    MOVL    12(RSTPTR), DSTPTR  ; 9284
                                                    04  AE  04  A6  9E 0002C   MOVAB   4(R6), 4(SP)        ; 9285
                                06      04  BE          08      18  ED 00031   CMPZV   #24, #8, @4(SP), #6   ;
                                                            12  13 00037   BEQL    2$                      ;
                                                        07  A4  9F 00039   PUSHAB  7(DSTPTR)               ; 9287
                                                        01  DD 0003C   PUSHL   #1                         ;
                                                000281A8    8F  DD 0003E   PUSHL   #164264                 ;
                                                    00000000G   00  03  FB 00044   CALLS   #3, LIB$SIGNAL  ;
                                                            1C  AE  9F 0004B 2$:    PUSHAB  TYPEID          ; 9289
                                                            24  AE  9F 0004E   PUSHAB  FCODE               ;
                                                            52  DD 00051   PUSHL   RSTPTR                  ;
                                                    00000000G   00  03  FB 00053   CALLS   #3, DBG$STA_SYMTYPE ;
                                                                0D      20  AE  D1 0005A   CMPL    FCODE, #13  ; 9290
                                                                        12  13 0005E   BEQL    3$          ;
                                                                    07  A4  9F 00060   PUSHAB  7(DSTPTR)   ; 9292
                                                                    01  DD 00063   PUSHL   #1             ;
                                                            000281A8    8F  DD 00065   PUSHL   #164264     ;
                                                                00000000G   00  03  FB 00068   CALLS   #3, LIB$SIGNAL ;
                                58      05  A4                  03      00  EF 00072 3$:    EXTZV   #0, #3, 5(DSTPTR), STRUC ; 9300
                                                                            12  12 00078   BNEQ    4$      ; 9301
                                                                    07  A4  9F 0007A   PUSHAB  7(DSTPTR)   ; 9303
```

```
                        01 DD 0007D          PUSHL   #1                                      ;
                000281A8    8F DD 0007F          PUSHL   #164264                                 ;
00000000G   00              03 FB 00085          CALLS   #3, LIB$SIGNAL                          ;
                        05  A4 95 0008C 4$:     TSTB    5(DSTPTR)                               ; 9310
                            1E 18 0008F          BGEQ    5$                                      ;
                            54 DD 00091          PUSHL   DSTPTR                                  ; 9314
                        08  A6 DD 00093          PUSHL   8(R6)                                   ;
                            0D DD 00096          PUSHL   #13                                     ; 9313
                    7E          06 7D 00098      MOVQ    #6, -(SP)                               ;
                            56 DD 0009B          PUSHL   R6                                      ;
                D88C    CF      06 FB 0009D      CALLS   #6, DBG$BUILD_PRIMARY_SUBNODE           ;
                    52      14  A6 D0 000A2      MOVL    20(R6), NODEPTR                         ; 9315
                  0A    A2      01 88 000A6      BISB2   #1, 10(NODEPTR)                         ; 9316
                    6E          01 D0 000AA      MOVL    #1, REF_FLAG                            ; 9317
                            02 11 000AD          BRB     6$                                      ; 9310
                            6E D4 000AF 5$:      CLRL    REF_FLAG                                ; 9320
                    52      18  A6 D0 000B1 6$:  MOVL    24(R6), NODEPTR                         ; 9326
                    53      28  A2 9E 000B5      MOVAB   40(R2), NODESUBPTR                      ; 9327
                    5B      08  A2 9E 000B9      MOVAB   8(NODEPTR), R11                         ; 9328
                  02    AB      01 88 000BD      BISB2   #1, 2(R11)                              ;
                            55 D4 000C1          CLRL    SUBSCR_COUNT                            ; 9336
00000000'       EF              01 D0 000C3      MOVL    #1, TERMINATOR_CODE                     ; 9337
                02 00000000'    EF D1 000CA 7$:  CMPL    TERMINATOR_CODE, #2                     ; 9338
                            03 12 000D1          BNEQ    8$                                      ;
                          01A1 31 000D3          BRW     29$                                     ;
                59 00000000'    EF D0 000D6 8$:  MOVL    CHARPTR, LA_PTR                         ; 9346
                20              69 91 000DD 9$:  CMPB    (LA_PTR), #32                           ; 9347
                            04 12 000E0          BNEQ    10$                                     ;
                            59 D6 000E2          INCL    LA_PTR                                  ;
                            F7 11 000E4          BRB     9$                                      ;
                        2A      69 91 000E6 10$: CMPB    (LA_PTR), #42                           ; 9348
                            03 13 000E9          BEQL    11$                                     ;
                          00B2 31 000EB          BRW     17$                                     ;
13                      6B      13 E0 000EE 11$: BBS     #19, (R11), 12$                         ; 9351
                            55 D5 000F2          TSTL    SUBSCR_COUNT                            ; 9352
                            0F 12 000F4          BNEQ    12$                                     ;
                        02      58 D1 000F6      CMPL    STRUC, #2                               ; 9353
                            17 13 000F9          BEQL    13$                                     ;
                        01      58 D1 000FB      CMPL    STRUC, #1                               ; 9354
                            12 13 000FE          BEQL    13$                                     ;
                        04      58 D1 00100      CMPL    STRUC, #4                               ; 9355
                            0D 13 00103          BEQL    13$                                     ;
                000028F08       8F DD 00105 12$: PUSHL   #167688                                 ; 9357
        00000000G   00              01 FB 0010B  CALLS   #1, LIB$SIGNAL                          ;
        00000000'   EF          01  A9 9E 00112 13$: MOVAB   1(R9), CHARPTR                      ; 9359
                            55 D6 0011A          INCL    SUBSCR_COUNT                            ; 9360
                            7E D4 0011C          CLRL    -(SP)                                   ; 9368
                00000000'       EF DD 0011E      PUSHL   SUBSCRIPT_TERM_TBL                      ; 9369
                            7E 7C 00124          CLRQ    -(SP)                                   ; 9368
                E235    CF          04 FB 00126  CALLS   #4, DBG$LEXICAL_SCANNER                 ;
                  18    AE          50 D0 0012B  MOVL    R0, TOKEN                               ;
                50 00000000'    EF 9E 0012F      MOVAB   TERMINATOR_TOKEN, R0                    ; 9370
                50          18  AE D1 00136      CMPL    TOKEN, R0                               ;
                            1E 13 0013A          BEQL    14$                                     ;
                  24    AE          01 90 0013C  MOVB    #1, ASCIC_STRING                        ; 9375
                  25    AE 00000000'  FF 90 00140  MOVB    @CHARPTR, ASCIC_STRING+1            ; 9376
                            24 AE 9F 00148       PUSHAB  ASCIC_STRING                            ; 9377
```

```
                                01   DD 0014B          PUSHL   #1                                      ;
                    000289E2    8F   DD 0014D          PUSHL   #166370                                 ;
00000000G     00                03   FB 00153          CALLS   #3, LIB$SIGNAL                          ;
                    03 00000000' EF   D1 0015A 14$:     CMPL    TERMINATOR_CODE, #3                     ; 9379
                                0D   12 00161          BNEQ    15$                                     ;
                    00028F08    8F   DD 00163          PUSHL   #167688                                 ; 9381
00000000G     00                01   FB 00169          CALLS   #1, LIB$SIGNAL                          ;
                    00000000'   EF   D5 00170 15$:     TSTL    TERMINATOR_CODE                         ; 9382
                                0D   12 00176          BNEQ    16$                                     ;
                    00028E90    8F   DD 00178          PUSHL   #167568                                 ; 9384
00000000G     00                01   FB 0017E          CALLS   #1, LIB$SIGNAL                          ;
00000000'     EF 00000000'      EF   C0 00185 16$:     ADDL2   TERMINATOR_LENGTH, CHARPTR              ; 9385
          02  AB                08   88 00190          BISB2   #8, 2(R11)                              ; 9390
              5A        08      A3   D0 00194          MOVL    8(NODESUBPTR), LOW_RANGE_VAL            ; 9391
          28  AE        0C      A3   D0 00198          MOVL    12(NODESUBPTR), SUBVECTOR               ; 9392
                      FF2A      31   00 0019D          BRW     7$                                      ; 9348
          10  AE 00000000'      EF   D0 001A0 17$:     MOVL    EXPRESSION_RADIX, SAVED_RADIX           ; 9404
00000000'     EF                0A   D0 001A8          MOVL    #10, EXPRESSION_RADIX                   ; 9405
                    00000000'   EF   DD 001AF          PUSHL   SUBSCRIPT_TERM_TBL                      ; 9406
                                7E   D4 001B5          CLRL    -(SP)                                   ;
         DCFE   CF              02   FB 001B7          CALLS   #2, DBG$EXPRESSION_PARSER               ;
                57              50   D0 001BC          MOVL    R0, VALPTR                              ;
00000000'     EF        10      AE   D0 001BF          MOVL    SAVED_RADIX, EXPRESSION_RADIX           ; 9407
                    00000000'   EF   D5 001C7          TSTL    TERMINATOR_CODE                         ; 9414
                                0D   12 001CD          BNEQ    18$                                     ;
                    00028E90    8F   DD 001CF          PUSHL   #167568                                 ;
00000000G     00                01   FB 001D5          CALLS   #1, LIB$SIGNAL                          ;
00000000'     EF 00000000'      EF   C0 001DC 18$:     ADDL2   TERMINATOR_LENGTH, CHARPTR              ; 9415
              0E        06      A7   91 001E7          CMPB    6(VALPTR), #14                          ; 9422
                                38   12 001EB          BNEQ    23$                                     ;
     11         6B              13   E1 001ED          BBC     #19, (R11), 19$                         ; 9425
                                55   D5 001F1          TSTL    SUBSCR_COUNT                            ;
                                0D   12 001F3          BNEQ    19$                                     ;
                    00028F08    8F   DD 001F5          PUSHL   #167688                                 ; 9427
00000000G     00                01   FB 001FB          CALLS   #1, LIB$SIGNAL                          ;
          08  AE        20      A7   9E 00202 19$:     MOVAB   32(R7), PTR                             ; 9429
          14  AE        08      BE   D0 00207          MOVL    @PTR, COUNT                             ; 9430
                                50   D4 0020C          CLRL    I                                       ; 9431
                                0E   11 0020E          BRB     22$                                     ;
              05                55   D1 00210 20$:     CMPL    SUBSCR_COUNT, #5                        ; 9433
                                07   18 00213          BGEQ    21$                                     ;
          28 AE45       08 BE40   D0 00215          MOVL    @PTR[I], SUBVECTOR[SUBSCR_COUNT]        ;
                                55   D6 0021C 21$:     INCL    SUBSCR_COUNT                            ; 9434
     ED         50        14      AE   F3 0021E 22$:     AOBLEQ  COUNT, I, 20$                           ; 9431
                                4F   11 00223          BRB     28$                                     ; 9422
                                57   DD 00225 23$:     PUSHL   VALPTR                                  ; 9450
          FB33  CF              01   FB 00227          CALLS   #1, CONVERT_TO_INTEGER                  ;
          0C    AE              50   D0 0022C          MOVL    R0, VALUE                               ;
                    03 00000000' EF   D1 00230          CMPL    TERMINATOR_CODE, #3                     ; 9459
                                2E   12 00237          BNEQ    26$                                     ;
     13         6B              13   E0 00239          BBS     #19, (R11), 24$                         ; 9462
                                55   D5 0023D          TSTL    SUBSCR_COUNT                            ; 9463
                                0F   12 0023F          BNEQ    24$                                     ;
              02                58   D1 00241          CMPL    STRUC, #2                               ; 9464
                                17   13 00244          BEQL    25$                                     ;
              01                58   D1 00246          CMPL    STRUC, #1                               ; 9465
                                12   13 00249          BEQL    25$                                     ;
```

```
                                    04              58  D1 0024B          CMPL    STRUC, #4                                   ; 9466
                                                    0D  13 0024E          BEQL    25$                                         ;
                                00028F08            8F  DD 00250 24$:     PUSHL   #167688                                     ; 9468
                        00000000G   00              01  FB 00256          CALLS   #1, LIB$SIGNAL                              ;
                                    02              08  88 0025D 25$:     BISB2   #8, 2(R11)                                  ; 9470
                                    5A          0C  AE  D0 00261          MOVL    VALUE, LOW_RANGE_VAL                        ; 9471
                                                    0D  11 00265          BRB     28$                                         ; 9459
                                    05              55  D1 00267 26$:     CMPL    SUBSCR_COUNT, #5                            ; 9483
                                                    06  18 0026A          BGEQ    27$                                         ;
                                28 AE45         0C  AE  D0 0026C          MOVL    VALUE, SUBVECTOR[SUBSCR_COUNT]              ; 9485
                                                    55  D6 00272 27$:     INCL    SUBSCR_COUNT                                ; 9487
                                                FE53    31 00274 28$:     BRW     7$                                          ; 9338
                                    04              58  CF 00277 29$:     CASEL   STRUC, #0, #4                               ; 9500
                00A9        001A            0036        000A      0027B 30$:     .WORD   31$-30$,-                           ;
                                                        01F7      00283             34$-30$,-                                 ;
                                                                                    32$-30$,-                                 ;
                                                                                    42$-30$,-                                 ;
                                                                                    62$-30$                                   ;
                                00028328            8F  DD 00285 31$:     PUSHL   #164650                                     ; 9821
                        00000000G   00              01  FB 0028B          CALLS   #1, LIB$SIGNAL                              ;
                                                037E    31 00292          BRW     85$                                         ;
                                                    55  D5 00295 32$:     TSTL    SUBSCR_COUNT                                ; 9512
                                                    11  14 00297          BGTR    33$                                         ;
                                                    01  DD 00299          PUSHL   #1                                          ;
                                                    01  DD 0029B          PUSHL   #1                                          ;
                                00028EA0            8F  DD 0029D          PUSHL   #167584                                     ;
                        00000000G   00              03  FB 002A3          CALLS   #3, LIB$SIGNAL                              ;
                                    01              55  D1 002AA 33$:     CMPL    SUBSCR_COUNT, #1                            ; 9513
                                                    1C  14 002AD          BGTR    36$                                         ;
                                                    2B  11 002AF          BRB     37$                                         ; 9518
                                                    55  D5 002B1 34$:     TSTL    SUBSCR_COUNT                                ; 9556
                                                    11  14 002B3          BGTR    35$                                         ;
                                                    01  DD 002B5          PUSHL   #1                                          ;
                                                    01  DD 002B7          PUSHL   #1                                          ;
                                00028EA0            8F  DD 002B9          PUSHL   #167584                                     ;
                        00000000G   00              03  FB 002BF          CALLS   #3, LIB$SIGNAL                              ;
                                    01              55  D1 002C6 35$:     CMPL    SUBSCR_COUNT, #1                            ; 9557
                                                    11  15 002C9          BLEQ    37$                                         ;
                                                    01  DD 002CB 36$:     PUSHL   #1                                          ;
                                                    01  DD 002CD          PUSHL   #1                                          ;
                                00028EB0            8F  DD 002CF          PUSHL   #167600                                     ;
                        00000000G   00              03  FB 002D5          CALLS   #3, LIB$SIGNAL                              ;
                                    42              6E  E8 002DC 37$:     BLBS    REF_FLAG, 41$                               ; 9562
                                    50          28  AE  D0 002DF          MOVL    SUBVECTOR, R0                               ; 9565
                                                    06  19 002E3          BLSS    38$                                         ;
                                06  A4              50  D1 002E5          CMPL    R0, 6(DSTPTR)                               ; 9566
                                                    14  19 002E9          BLSS    39$                                         ;
                                                    50  DD 002EB 38$:     PUSHL   R0                                          ; 9569
                                            06      A4  DD 002ED          PUSHL   6(DSTPTR)                                   ;
                                                    02  DD 002F0          PUSHL   #2                                          ; 9568
                                0002807B            8F  DD 002F2          PUSHL   #163963                                     ;
                        00000000G   00              04  FB 002F8          CALLS   #4, LIB$SIGNAL                              ;
                    1E              6B              13  E1 002FF 39$:     BBC     #19, (R11), 41$                             ; 9575
                                                    5A  D5 00303          TSTL    LOW_RANGE_VAL                               ; 9576
                                                    06  19 00305          BLSS    40$                                         ;
                                06  A4              5A  D1 00307          CMPL    LOW_RANGE_VAL, 6(DSTPTR)                    ; 9577
                                                    14  19 0030B          BLSS    41$                                         ;
```

```
                                  5A  DD 0030D 40$:   PUSHL   LOW_RANGE_VAL                      ; 9580
                            06    A4  DD 0030F        PUSHL   6(DSTPTR)                          ;
                                  02  DD 00312        PUSHL   #2                                 ; 9579
                      0002807B    8F  DD 00314        PUSHL   #163963                            ;
             00000000G   00       04  FB 0031A        CALLS   #4, LIB$SIGNAL                     ;
                                0143  31 00321 41$:   BRW     61$                                ; 9586
 50       0A    A4       04       00  EF 00324 42$:   EXTZV   #0, #4, 10(DSTPTR), STRIDE         ; 9601
                   04    A3       50  D0 0032A        MOVL    STRIDE, 4(NODESUBPTR)              ; 9602
          0C    A3 06    A4       01  C3 0032E        SUBL3   #1, 6(DSTPTR), 12(NODESUBPTR)      ; 9604
                   1C    A2       50  B0 00334        MOVW    STRIDE, 28(NODEPTR)                ; 9605
                         51  18   A2  9E 00338        MOVAB   24(NODEPTR), R1                    ; 9610
                         01       50  D1 0033C        CMPL    STRIDE, #1                         ; 9606
                                  10  12 0033F        BNEQ    44$                                ;
                         06  34   AE  E9 00341        BLBC    SUBVECTOR+12, 43$                  ; 9608
                   02    A1       06  90 00345        MOVB    #6, 2(R1)                          ; 9610
                                  42  11 00349        BRB     51$                                ;
                   02    A1       02  90 0034B 43$:   MOVB    #2, 2(R1)                          ; 9612
                                  3C  11 0034F        BRB     51$                                ; 9608
                         02       50  D1 00351 44$:   CMPL    STRIDE, #2                         ; 9613
                                  10  12 00354        BNEQ    46$                                ;
                         06  34   AE  E9 00356        BLBC    SUBVECTOR+12, 45$                  ; 9615
                   02    A1       07  90 0035A        MOVB    #7, 2(R1)                          ; 9617
                                  2D  11 0035E        BRB     51$                                ;
                   02    A1       03  90 00360 45$:   MOVB    #3, 2(R1)                          ; 9619
                                  27  11 00364        BRB     51$                                ; 9615
                         04       50  D1 00366 46$:   CMPL    STRIDE, #4                         ; 9620
                                  10  12 00369        BNEQ    48$                                ;
                         06  34   AE  E9 0036B        BLBC    SUBVECTOR+12, 47$                  ; 9622
                   02    A1       08  90 0036F        MOVB    #8, 2(R1)                          ; 9624
                                  18  11 00373        BRB     51$                                ;
                   02    A1       04  90 00375 47$:   MOVB    #4, 2(R1)                          ; 9626
                                  12  11 00379        BRB     51$                                ; 9622
                         06  34   AE  E9 0037B 48$:   BLBC    SUBVECTOR+12, 49$                  ; 9629
                   02    A1       01  90 0037F        MOVB    #1, 2(R1)                          ; 9631
                                  04  11 00383        BRB     50$                                ;
                   02    A1       22  90 00385 49$:   MOVB    #34, 2(R1)                         ; 9633
                   1C    A2       08  A4 00389 50$:   MULW2   #8, 28(NODEPTR)                    ; 9634
                         04       55  D1 0038D 51$:   CMPL    SUBSCR_COUNT, #4                   ; 9639
                                  11  18 00390        BGEQ    52$                                ;
                                  04  DD 00392        PUSHL   #4                                 ;
                                  01  DD 00394        PUSHL   #1                                 ;
                      00028EA0    8F  DD 00396        PUSHL   #167584                            ;
             00000000G   00       03  FB 0039C        CALLS   #3, LIB$SIGNAL                     ;
                         04       55  D1 003A3 52$:   CMPL    SUBSCR_COUNT, #4                   ; 9640
                                  11  15 003A6        BLEQ    53$                                ;
                                  04  DD 003A8        PUSHL   #4                                 ;
                                  01  DD 003AA        PUSHL   #1                                 ;
                      00028EB0    8F  DD 003AC        PUSHL   #167600                            ;
             00000000G   00       03  FB 003B2        CALLS   #3, LIB$SIGNAL                     ;
                         20       6E  E8 003B9 53$:   BLBS    REF_FLAG, 55$                      ; 9645
                         50  28   AE  D0 003BC        MOVL    SUBVECTOR, R0                      ; 9647
                                  06  19 003C0        BLSS    54$                                ;
                   06    A4       50  D1 003C2        CMPL    R0, 6(DSTPTR)                      ; 9648
                                  14  19 003C6        BLSS    55$                                ;
                                  50  DD 003C8 54$:   PUSHL   R0                                 ; 9651
                            06    A4  DD 003CA        PUSHL   6(DSTPTR)                          ;
                                  02  DD 003CD        PUSHL   #2                                 ; 9650
```

```
                        0002807B  8F  DD 003CF          PUSHL   #163963                                  ;
00000000G               00        04  FB 003D5          CALLS   #4, LIB$SIGNAL                          ;
FFFF8000                8F    2C  AE  D1 003DC 55$:     CMPL    SUBVECTOR+4, #-32768                    ; 9653
                              0A  19 003E4              BLSS    56$                                     ;
00007FFF                8F    2C  AE  D1 003E6          CMPL    SUBVECTOR+4, #32767                     ; 9654
                              12  15 003EE              BLEQ    57$                                     ;
                          2C      AE  DD 003F0 56$:     PUSHL   SUBVECTOR+4                             ; 9656
                              01  DD 003F3              PUSHL   #1                                      ;
                        000280F0  8F  DD 003F5          PUSHL   #164080                                 ;
00000000G               00        03  FB 003FB          CALLS   #3, LIB$SIGNAL                          ;
                        58    30  AE  D0 00402 57$:     MOVL    SUBVECTOR+8, R8                         ; 9658
                              11  18 00406              BGEQ    58$                                     ;
                              58  DD 00408              PUSHL   R8                                      ; 9660
                              01  DD 0040A              PUSHL   #1                                      ;
                        00028EE8  8F  DD 0040C          PUSHL   #167656                                 ;
00000000G               00        03  FB 00412          CALLS   #3, LIB$SIGNAL                          ;
                        20        58  D1 00419 58$:     CMPL    R8, #32                                 ; 9662
                              11  15 0041C              BLEQ    59$                                     ;
                    30  AE        20  D0 0041E          MOVL    #32, SUBVECTOR+8                        ; 9665
                        00028073  8F  DD 00422          PUSHL   #163955                                 ; 9666
00000000G               00        01  FB 00428          CALLS   #1, LIB$SIGNAL                          ;
                        58    34  AE  D0 0042F 59$:     MOVL    SUBVECTOR+12, R8                        ; 9669
                              16  13 00433              BEQL    60$                                     ;
                        01        58  D1 00435          CMPL    R8, #1                                  ;
                              11  13 00438              BEQL    60$                                     ;
                              58  DD 0043A              PUSHL   R8                                      ; 9671
                              01  DD 0043C              PUSHL   #1                                      ;
                        00028140  8F  DD 0043E          PUSHL   #164160                                 ;
                00000000G       00    03  FB 00444      CALLS   #3, LIB$SIGNAL                          ;
                          04    BE    10  88 0044B 60$:     BISB2   #16, @4(SP)                         ; 9676
                          04    BE    02  88 0044F          BISB2   #2, @4(SP)                          ; 9677
                          04    BE    04  88 00453          BISB2   #4, @4(SP)                          ; 9678
                          10    A6  2C  AE  B0 00457        MOVW    SUBVECTOR+4, 16(R6)                 ; 9679
                          12    A6  30  AE  B0 0045C        MOVW    SUBVECTOR+8, 18(R6)                 ; 9680
04  BE          01              03        58  F0 00461      INSV    R8, #3, #1, @4(SP)                  ; 9681
                          1F    A2        01  90 00467 61$:     MOVB    #1, 31(NODEPTR)                 ; 9682
                              63      28  AE  D0 0046B        MOVL    SUBVECTOR, (NODESUBPTR)             ; 9683
                                    01A1  31  0046F          BRW     85$                                 ; 9500
                              50      0E  A4  9A 00472 62$:     MOVZBL  14(DSTPTR), STRIDE                ; 9699
                          18  A3          50  D0 00476          MOVL    STRIDE, 24(NODESUBPTR)            ; 9700
                    20    A3    0A  A4        01  C3 0047A      SUBL3   #1, 10(DSTPTR), 32(NODESUBPTR)    ; 9702
                          1C    A2        50  B0 00480        MOVW    STRIDE, 28(NODEPTR)                 ; 9703
                              51      18  A2  9E 00484        MOVAB   24(NODEPTR), R1                     ; 9708
                              01      50  D1 00488          CMPL    STRIDE, #1                          ; 9704
                                      10  12 0048B          BNEQ    64$                                 ;
                              06      38  AE  E9 0048D      BLBC    SUBVECTOR+16, 63$                   ; 9706
                          02  A1          06  90 00491          MOVB    #6, 2(R1)                         ; 9708
                                      42  11 00495          BRB     71$                                 ;
                          02  A1          02  90 00497 63$:     MOVB    #2, 2(R1)                         ; 9710
                                      3C  11 0049B          BRB     71$                                 ; 9706
                              02      50  D1 0049D 64$:     CMPL    STRIDE, #2                          ; 9711
                                      10  12 004A0          BNEQ    66$                                 ;
                              06      38  AE  E9 004A2      BLBC    SUBVECTOR+16, 65$                   ; 9713
                          02  A1          07  90 004A6          MOVB    #7, 2(R1)                         ; 9715
                                      2D  11 004AA          BRB     71$                                 ;
                          02  A1          03  90 004AC 65$:     MOVB    #3, 2(R1)                         ; 9717
                                      27  11 004B0          BRB     71$                                 ; 9713
```

```
                        04              50  D1 004B2 66$:    CMPL    STRIDE, #4                              ; 9718
                                        10  12 004B5         BNEQ    68$                                     ;
                        06        38    AE  E9 004B7         BLBC    SUBVECTOR+16, 67$                       ; 9720
                    02  A1              08  90 004BB         MOVB    #8, 2(R1)                               ; 9722
                                        18  11 004BF         BRB     71$                                     ;
                    02  A1              04  90 004C1 67$:    MOVB    #4, 2(R1)                               ; 9724
                                        12  11 004C5         BRB     71$                                     ; 9720
                        06        38    AE  E9 004C7 68$:    BLBC    SUBVECTOR+16, 69$                       ; 9727
                    02  A1              01  90 004CB         MOVB    #1, 2(R1)                               ; 9729
                                        04  11 004CF         BRB     70$                                     ;
                    02  A1              22  90 004D1 69$:    MOVB    #34, 2(R1)                              ; 9731
                    1C  A2              08  A4 004D5 70$:    MULW2   #8, 28(NODEPTR)                         ; 9732
                        05              55  D1 004D9 71$:    CMPL    SUBSCR_COUNT, #5                        ; 9737
                                        11  18 004DC         BGEQ    72$                                     ;
                                        05  DD 004DE         PUSHL   #5                                      ;
                                        01  DD 004E0         PUSHL   #1                                      ;
                            00028EA0    8F  DD 004E2         PUSHL   #167584                                 ;
                00000000G   00          03  FB 004E8         CALLS   #3, LIB$SIGNAL                          ;
                        05              55  D1 004EF 72$:    CMPL    SUBSCR_COUNT, #5                        ; 9738
                                        11  15 004F2         BLEQ    73$                                     ;
                                        05  DD 004F4         PUSHL   #5                                      ;
                                        01  DD 004F6         PUSHL   #1                                      ;
                            00028EB0    8F  DD 004F8         PUSHL   #167600                                 ;
                00000000G   00          03  FB 004FE         CALLS   #3, LIB$SIGNAL                          ;
                        40              6E  E8 00505 73$:    BLBS    REF_FLAG, 77$                           ; 9743
                        50        28    AE  D0 00508         MOVL    SUBVECTOR, R0                           ; 9746
                                        06  19 0050C         BLSS    74$                                     ;
                    06  A4              50  D1 0050E         CMPL    R0, 6(DSTPTR)                           ; 9747
                                        14  19 00512         BLSS    75$                                     ;
                                        50  DD 00514 74$:    PUSHL   R0                                      ; 9750
                                06      A4  DD 00516         PUSHL   6(DSTPTR)                               ;
                                        02  DD 00519         PUSHL   #2                                      ; 9749
                            0002807B    8F  DD 0051B         PUSHL   #163963                                 ;
                00000000G   00          04  FB 00521         CALLS   #4, LIB$SIGNAL                          ;
                        50        2C    AE  D0 00528 75$:    MOVL    SUBVECTOR+4, R0                         ; 9752
                                        06  19 0052C         BLSS    76$                                     ;
                    0A  A4              50  D1 0052E         CMPL    R0, 10(DSTPTR)                          ; 9753
                                        14  19 00532         BLSS    77$                                     ;
                                        50  DD 00534 76$:    PUSHL   R0                                      ; 9756
                                0A      A4  DD 00536         PUSHL   10(DSTPTR)                              ;
                                        02  DD 00539         PUSHL   #2                                      ; 9755
                            0002807B    8F  DD 0053B         PUSHL   #163963                                 ;
                00000000G   00          04  FB 00541         CALLS   #4, LIB$SIGNAL                          ;
                FFFF8000    8F    30    AE  D1 00548 77$:    CMPL    SUBVECTOR+8, #-32768                    ; 9759
                                        0A  19 00550         BLSS    78$                                     ;
                00007FFF    8F    30    AE  D1 00552         CMPL    SUBVECTOR+8, #32767                     ; 9760
                                        12  15 0055A         BLEQ    79$                                     ;
                                30      AE  DD 0055C 78$:    PUSHL   SUBVECTOR+8                             ; 9762
                                        01  DD 0055F         PUSHL   #1                                      ;
                            000280F0    8F  DD 00561         PUSHL   #164080                                 ;
                00000000G   00          03  FB 00567         CALLS   #3, LIB$SIGNAL                          ;
                        58        34    AE  D0 0056E 79$:    MOVL    SUBVECTOR+12, R8                        ; 9764
                                        11  18 00572         BGEQ    80$                                     ;
                                        58  DD 00574         PUSHL   R8                                      ; 9766
                                        01  DD 00576         PUSHL   #1                                      ;
                            00028EE8    8F  DD 00578         PUSHL   #167656                                 ;
                00000000G   00          03  FB 0057E         CALLS   #3, LIB$SIGNAL                          ;
```

```
                        20              58 D1 00585 80$:    CMPL    R8, #32                         ; 9768
                                        11 15 00588         BLEQ    81$                             ;
                     34 AE              20 D0 0058A         MOVL    #32, SUBVECTOR+12               ; 9771
                        00028073        8F DD 0058E         PUSHL   #163955                         ; 9772
                00000000G 00            01 FB 00594         CALLS   #1, LIB$SIGNAL                  ;
                        58      38 AE   D0 0059B 81$:       MOVL    SUBVECTOR+16, R8                ; 9775
                                        16 13 0059F         BEQL    82$                             ;
                        01              58 D1 005A1         CMPL    R8, #1                          ;
                                        11 13 005A4         BEQL    82$                             ;
                                        58 DD 005A6         PUSHL   R8                              ; 9777
                                        01 DD 005A8         PUSHL   #1                              ;
                        00028140        8F DD 005AA         PUSHL   #164160                         ;
                00000000G 00            03 FB 005B0         CALLS   #3, LIB$SIGNAL                  ;
                        22              6E E8 005B7 82$:    BLBS    REF_FLAG, 84$                   ; 9783
             1E         6B              13 E1 005BA         BBC     #19, (R11), 84$                 ; 9785
                                        5A D5 005BE         TSTL    LOW_RANGE_VAL                   ; 9786
                                        06 19 005C0         BLSS    83$                             ;
                     06 A4              5A D1 005C2         CMPL    LOW_RANGE_VAL, 6(DSTPTR)        ; 9787
                                        14 19 005C6         BLSS    84$                             ;
                                        5A DD 005C8 83$:    PUSHL   LOW_RANGE_VAL                   ; 9790
                           06 A4        DD 005CA            PUSHL   6(DSTPTR)                       ;
                                        02 DD 005CD         PUSHL   #2                              ; 9789
                        0002807B        8F DD 005CF         PUSHL   #163963                         ;
                00000000G 00            04 FB 005D5         CALLS   #4, LIB$SIGNAL                  ;
                     04 BE              10 88 005DC 84$:    BISB2   #16, @4(SP)                     ; 9795
                     04 BE              02 88 005E0         BISB2   #2, @4(SP)                      ; 9796
                     04 BE              04 88 005E4         BISB2   #4, @4(SP)                      ; 9797
                     10 A6      30 AE   B0 005E8            MOVW    SUBVECTOR+8, 16(R6)             ; 9798
                     12 A6      34 AE   B0 005ED            MOVW    SUBVECTOR+12, 18(R6)            ; 9799
          04 BE         01              03 58 F0 005F2      INSV    R8, #3, #1, @4(SP)              ; 9800
                     1F A2              02 90 005F8         MOVB    #2, 31(NODEPTR)                 ; 9801
                        63      28 AE   D0 005FC            MOVL    SUBVECTOR, (NODESUBPTR)         ; 9802
                     14 A3      2C AE   D0 00600            MOVL    SUBVECTOR+4, 20(NODESUBPTR)     ; 9803
                  2E            6B      13 E1 00605         BBC     #19, (R11), 87$                 ; 9808
                     1C A3      2C AE   D0 00609            MOVL    SUBVECTOR+4, 28(NODESUBPTR)     ; 9811
                     20 A3      2C AE   D0 0060E            MOVL    SUBVECTOR+4, 32(NODESUBPTR)     ; 9812
                  20            6B      13 E1 00613 85$:    BBC     #19, (R11), 87$                 ; 9830
                     28 AE              5A D1 00617         CMPL    LOW_RANGE_VAL, SUBVECTOR        ; 9833
                                        0D 15 0061B         BLEQ    86$                             ;
                           00028F08     8F DD 0061D         PUSHL   #167688                         ;
                   00000000G 00         01 FB 00623         CALLS   #1, LIB$SIGNAL                  ;
                        63              5A D0 0062A 86$:    MOVL    LOW_RANGE_VAL, (NODESUBPTR)     ; 9834
                     08 A3              5A D0 0062D         MOVL    LOW_RANGE_VAL, 8(NODESUBPTR)    ; 9835
                     0C A3      28 AE   D0 00631            MOVL    SUBVECTOR, 12(NODESUBPTR)       ; 9836
                                        04 00636            RET                                     ; 9832
                                        7E D4 00637 87$:    CLRL    -(SP)                           ; 9845
                           24 A2        DD 00639            PUSHL   36(NODEPTR)                     ; 9846
                                        02 DD 0063C         PUSHL   #2                              ; 9845
                        7E              06 7D 0063E         MOVQ    #6, -(SP)                       ;
                                        56 DD 00641         PUSHL   R6                              ;
                   D2E6 CF              06 FB 00643         CALLS   #6, DBG$BUILD_PRIMARY_SUBNODE   ;
                                        04 00648            RET                                     ; 9848

; Routine Size:  1609 bytes,    Routine Base:  DBG$CODE + 273E
```

```
;  9753     9849  1 ROUTINE GET_DEREFERENCE (PRIMPTR): NOVALUE =
;  9754     9850  1 !
;  9755     9851  1 ! FUNCTION
;  9756     9852  1 !       This routine is called upon seeing the dereference operator, e.g.,
;  9757     9853  1 !       the ^ in a PASCAL primary such as A^.
;  9758     9854  1 !
;  9759     9855  1 !       If the object being dereferenced is a pointer or a file variable, then
;  9760     9856  1 !       this routine lights a bit in the current primary subnode which
;  9761     9857  1 !       indicates that the dereference is taking place. It then calls
;  9762     9858  1 !       DBG$BUILD_PRIMARY_SUBNODE to append a new subnode. The type
;  9763     9859  1 !       information in the new subnode reflects the type of the object
;  9764     9860  1 !       being pointed to; or in the case of file variables, the type
;  9765     9861  1 !       of the objects in the file.
;  9766     9862  1 !
;  9767     9863  1 ! INPUTS
;  9768     9864  1 !       PRIMPTR         - A pointer to the Primary Descriptor currently
;  9769     9865  1 !                         being constructed by DBG$PRIMARY_PARSER.
;  9770     9866  1 !
;  9771     9867  1 ! OUTPUTS
;  9772     9868  1 !       The Primary Descriptor pointed to by PRIMPTR is modified.
;  9773     9869  1 !
;  9774     9870  2     BEGIN
;  9775     9871  2
;  9776     9872  2     MAP
;  9777     9873  2         PRIMPTR: REF DBG$PRIMARY;
;  9778     9874  2
;  9779     9875  2     LOCAL
;  9780     9876  2         FCODE,                          ! Local variable holding fcode info
;  9781     9877  2         JUNK,                           ! Dummy output parameter
;  9782     9878  2         NODEPTR: REF DBG$PRIM_NODE,     ! Points to a Primary Sub-node
;  9783     9879  2         TYPEID;                         ! Pointer to a RST type entry
;  9784     9880  2
;  9785     9881  2     DBG$GL_CURRENT_PRIMARY = .PRIMPTR;
;  9786     9882  2
;  9787     9883  2
;  9788     9884  2     ! Check that the object being dereferenced is actually a pointer.
;  9789     9885  2     !
;  9790     9886  2     IF  .PRIMPTR[DBG$B_DHDR_FCODE] NEQ RST$K_TYPE_TPTR
;  9791     9887  2     AND .PRIMPTR[DBG$B_DHDR_FCODE] NEQ RST$K_TYPE_FILE
;  9792     9888  2     THEN
;  9793     9889  2         SIGNAL(DBG$_NOTPTR);
;  9794     9890  2
;  9795     9891  2
;  9796     9892  2     ! Obtain a pointer to the bottom level sub-node by following the
;  9797     9893  2     ! back-pointer. Light the EVAL bit in this subnode,
;  9798     9894  2     ! which indicates that pointer dereferencing is
;  9799     9895  2     ! taking place.
;  9800     9896  2     ! Then, obtain the pointer to the RST type entry for the object being
;  9801     9897  2     ! dereferenced.
;  9802     9898  2     !
;  9803     9899  2     NODEPTR = .PRIMPTR [DBG$L_PRIM_BLINK];
;  9804     9900  2     NODEPTR [DBG$V_PNODE_EVAL] = TRUE;
;  9805     9901  2     TYPEID = .NODEPTR [DBG$L_PNODE_TYPEID];
;  9806     9902  2
;  9807     9903  2
;  9808     9904  2     ! From this typeid, get the typeid for the object being pointed to.
;  9809     9905  2     ! For pointer variables, use the routine that extracts the typeid
```

```
; 9810          9906  2     ! of the pointed-to object.
; 9811          9907  2     ! For file variables, use the routine that extracts the typeid of
; 9812          9908  2     ! objects in the file.
; 9813          9909  2     ! Then obtain the fcode from the typeid.
; 9814          9910  2     !
; 9815          9911  2     IF  .PRIMPTR[DBG$B_DHDR_FCODE] EQL RST$K_TYPE_TPTR
; 9816          9912  2     THEN
; 9817          9913  2         DBG$STA_TYP_TYPEDPTR (.TYPEID, TYPEID)
; 9818          9914  2     ELSE
; 9819          9915  2         DBG$STA_TYP_FILE (.TYPEID, JUNK, TYPEID);
; 9820          9916  2     FCODE = DBG$STA_TYPEFCODE (.TYPEID);
; 9821          9917  2
; 9822          9918  2
; 9823          9919  2     ! Append a new sub-node to the Primary Descriptor.
; 9824          9920  2     !
; 9825          9921  2     DBG$BUILD_PRIMARY_SUBNODE (.PRIMPTR, RST$K_DATA, 0, .FCODE, .TYPEID, 0);
; 9826          9922  2     RETURN;
; 9827          9923  1     END;


                                    0004 00000 GET_DEREFERENCE:
                                               .WORD   Save R2                                          ; 9849
                          5E            08  C2 00002          SUBL2   #8, SP                           ;
                          52      04    AC  D0 00005          MOVL    PRIMPTR, R2                      ; 9881
00000000G                 00            52  D0 00009          MOVL    R2, DBG$GL_CURRENT_PRIMARY       ;
                          06      06    A2  91 00010          CMPB    6(R2), #6                        ; 9886
                                        13  13 00014          BEQL    1$                               ;
                          0F      06    A2  91 00016          CMPB    6(R2), #15                       ; 9887
                                        0D  13 0001A          BEQL    1$                               ;
                              000287F0  8F  DD 0001C          PUSHL   #165872                          ; 9889
00000000G                 00            01  FB 00022          CALLS   #1, LIB$SIGNAL                   ;
                          50      18    A2  D0 00029  1$:     MOVL    24(R2), NODEPTR                  ; 9899
                    0A    A0            01  88 0002D          BISB2   #1, 10(NODEPTR)                  ; 9900
                          6E      0C    A0  D0 00031          MOVL    12(NODEPTR), TYPEID              ; 9901
                          06      06    A2  91 00035          CMPB    6(R2), #6                        ; 9911
                                        0E  12 00039          BNEQ    2$                               ;
                                        5E  DD 0003B          PUSHL   SP                               ; 9913
                                  04    AE  DD 0003D          PUSHL   TYPEID                           ;
00000000G                 00            02  FB 00040          CALLS   #2, DBG$STA_TYP_TYPEDPTR         ;
                                        0F  11 00047          BRB     3$                               ;
                                        5E  DD 00049  2$:     PUSHL   SP                               ; 9915
                                  08    AE  9F 0004B          PUSHAB  JUNK                             ;
                                  08    AE  DD 0004E          PUSHL   TYPEID                           ;
00000000G                 00            03  FB 00051          CALLS   #3, DBG$STA_TYP_FILE             ;
                                        6E  DD 00058  3$:     PUSHL   TYPEID                           ; 9916
00000000G                 00            01  FB 0005A          CALLS   #1, DBG$STA_TYPEFCODE            ;
                                        7E  D4 00061          CLRL    -(SP)                            ; 9921
                                  04    AE  DD 00063          PUSHL   TYPEID                           ;
                                        50  DD 00066          PUSHL   FCODE                            ;
                          7E            06  7D 00068          MOVQ    #6, -(SP)                        ;
                                        52  DD 0006B          PUSHL   R2                               ;
                    D273  CF            06  FB 0006D          CALLS   #6, DBG$BUILD_PRIMARY_SUBNODE    ;
                                            04 00072          RET                                      ; 9923
```

; Routine Size: 115 bytes,   Routine Base: DBG$CODE + 2D87

```
;  9829          9924  1 ROUTINE GET_FIELDREF (TOKEN): NOVALUE =
;  9830          9925  1 !
;  9831          9926  1 ! FUNCTION
;  9832          9927  1 !       This routine picks up the position, size, and (optionally) the
;  9833          9928  1 !       extension in a field reference (i.e., XX<pos,size,ext>.
;  9834          9929  1 !       DBG$EXPRESSION_PARSER is called to parse and evaluate each of
;  9835          9930  1 !       these values. The values are stored as integers in the
;  9836          9931  1 !       Operator Lexical Token entry.
;  9837          9932  1 !
;  9838          9933  1 !       This routine assumes that the opening angle bracket has already
;  9839          9934  1 !       been found and that the parse pointer points to the start of
;  9840          9935  1 !       the first expression in the field reference. When this routine
;  9841          9936  1 !       returns, the parse pointer is left at the first character after
;  9842          9937  1 !       the closing angle bracket.
;  9843          9938  1 !
;  9844          9939  1 ! INPUTS
;  9845          9940  1 !       TOKEN   - a pointer to the Operator Lexical Token entry for
;  9846          9941  1 !                 the "<" operator.
;  9847          9942  1 !
;  9848          9943  1 ! OUTPUTS
;  9849          9944  1 !       The Lexical Token pointed to by TOKEN is modified to include the
;  9850          9945  1 !       offset, size, and sign extension information.
;  9851          9946  1 !
;  9852          9947  1
;  9853          9948  2     BEGIN
;  9854          9949  2
;  9855          9950  2     MAP
;  9856          9951  2         TOKEN: REF TOKEN$ENTRY;          ! Pointer to the Lexical Token Entry
;  9857          9952  2                                          !      for the field reference operator
;  9858          9953  2
;  9859          9954  2     LOCAL
;  9860          9955  2         DECLTYPE: REF DBG$VALDESC,       ! Pointer to Value Descriptor
;  9861          9956  2                                          !      for one of the values
;  9862          9957  2                                          !      inside the angle brackets
;  9863          9958  2         SAVED_RADIX,                     ! Temporarily saved expression radix
;  9864          9959  2         VALUE,                           ! Value of position, size, or sign ext
;  9865          9960  2                                          !      field
;  9866          9961  2         VALPTR: REF DBG$VALDESC;         ! Pointer to position, size, or
;  9867          9962  2                                          !      extension value descriptor.
;  9868          9963  2
;  9869          9964  2
;  9870          9965  2
;  9871          9966  2     ! Loop through the expressions in this field reference. Each of these
;  9872          9967  2     ! is parsed and evaluated via a call to DBG$EXPRESSION_PARSER.
;  9873          9968  2     ! This returns a descriptor, and the type converter is then called
;  9874          9969  2     ! to convert the descriptor into an integer value.
;  9875          9970  2     ! The integer value is checked for being in an appropriate range and
;  9876          9971  2     ! then stored in the appropriate own variable.
;  9877          9972  2     !
;  9878          9973  2     INCR I FROM 1 TO 3 DO
;  9879          9974  3         BEGIN
;  9880          9975  3
;  9881          9976  3
;  9882          9977  3         ! Call the expression parser to pick up the next expression in the
;  9883          9978  3         ! field reference.  Note that we set the radix to decimal over this
;  9884          9979  3         ! call and then restore it.  Also note that the Expression Parser sets
;  9885          9980  3         ! TERMINATOR_CODE and TERMINATOR_LENGTH as a side-effect.
```

```
;  9886          9981  3          !
;  9887          9982  3          SAVED_RADIX = .EXPRESSION_RADIX;
;  9888          9983  3          EXPRESSION_RADIX = DBG$K_DECIMAL;
;  9889          9984  3          VALPTR = DBG$EXPRESSION_PARSER(FALSE, BIT_SELECT_TERM_TBL);
;  9890          9985  3          EXPRESSION_RADIX = .SAVED_RADIX;
;  9891          9986  3
;  9892          9987  3
;  9893          9988  3          ! Check the terminator code.  If there was no terminator (i.e., the
;  9894          9989  3          ! input line just ended), signal an error.  Otherwise we got a comma
;  9895          9990  3          ! or closing angle bracket and we increment CHARPTR to get past it.
;  9896          9991  3          !
;  9897          9992  3          IF .TERMINATOR_CODE EQL TOKEN$K_TERM_NONE THEN SIGNAL(DBG$_MISCLOSUB);
;  9898          9993  3          CHARPTR = .CHARPTR + .TERMINATOR_LENGTH;
;  9899          9994  3
;  9900          9995  3
;  9901          9996  3          ! Convert the value descriptor to an integer.
;  9902          9997  3          !
;  9903          9998  3          VALUE = CONVERT_TO_INTEGER (.VALPTR);
;  9904          9999  3
;  9905         10000  3
;  9906         10001  3                  ! If we are looking at the "position" field, it can be any value that
;  9907         10002  3                  ! fits in the PRIM_OFFSET field of the Primary Descriptor.  We do not
;  9908         10003  3                  ! allow any terminator other than comma in this case.
;  9909         10004  3                  !
;  9910         10005  3                  IF .I EQL 1
;  9911         10006  3                  THEN
;  9912         10007  4                      BEGIN
;  9913         10008  4                      IF .VALUE LSS %X'0000' OR .VALUE GTR %X'7FFF'
;  9914         10009  4                      THEN
;  9915         10010  4                          SIGNAL (DBG$_ILLPOSFLD, 1, .VALUE);                                    ;
;  9916         10011  4
;  9917         10012  4                      TOKEN [TOKEN$W_BIT_OFFSET] = .VALUE;
;  9918         10013  4                      IF .TERMINATOR_CODE EQL TOKEN$K_TERM_GTRTHAN
;  9919         10014  4                      THEN
;  9920         10015  4                          SIGNAL(DBG$_INVFLDREF);
;  9921         10016  4
;  9922         10017  3                      END;
;  9923         10018  3
;  9924         10019  3
;  9925         10020  3                  ! If we are looking at the size field, the value must be in the
;  9926         10021  3                  ! range 0 to 32 inclusive.  Here we exit the loop normally on a
;  9927         10022  3                  ! closing angle bracket, defaulting the sign extension to zero.
;  9928         10023  3                  !
;  9929         10024  3                  IF .I EQL 2
;  9930         10025  3                  THEN
;  9931         10026  4                      BEGIN
;  9932         10027  4                      IF .VALUE LSS 0 OR .VALUE GTR 32
;  9933         10028  4                      THEN
;  9934         10029  4                          SIGNAL (DBG$_ILLSIZFLD, 1, .VALUE);
;  9935         10030  4
;  9936         10031  4                      TOKEN [TOKEN$W_BIT_LENGTH] = .VALUE;
;  9937         10032  4                      IF .TERMINATOR_CODE EQL TOKEN$K_TERM_GTRTHAN THEN EXITLOOP;
;  9938         10033  3                      END;
;  9939         10034  3
;  9940         10035  3
;  9941         10036  3                  ! If we are looking at the sign extension field then the value
;  9942         10037  3                  ! must be either 0 or 1.  We also insist that the terminator must
```

```
;  9943     10038  3              ! be the closing angle bracket in this case.
;  9944     10039  3              !
;  9945     10040  3              IF .I EQL 3
;  9946     10041  3              THEN
;  9947     10042  4                  BEGIN
;  9948     10043  4                  IF .VALUE NEQ 0 AND .VALUE NEQ 1
;  9949     10044  4                  THEN
;  9950     10045  4                      SIGNAL (DBG$_ILLSIGEXT, 1, .VALUE);
;  9951     10046  4
;  9952     10047  4                  TOKEN [TOKEN$V_SGNEXT] = .VALUE;
;  9953     10048  4                  IF .TERMINATOR_CODE NEQ TOKEN$K_TERM_GTRTHAN
;  9954     10049  4                  THEN
;  9955     10050  4                      SIGNAL(DBG$_INVFLDREF);
;  9956     10051  4
;  9957     10052  4                  EXITLOOP;
;  9958     10053  3                  END;
;  9959     10054  3
;  9960     10055  2              END;                              ! End of INCR loop
;  9961     10056  2
;  9962     10057  2
;  9963     10058  2          ! All done. Return to the caller.
;  9964     10059  2          !
;  9965     10060  2          RETURN;
;  9966     10061  1          END;


                                  00FC 00000 GET_FIELDREF:
                                                  .WORD   Save R2,R3,R4,R5,R6,R7          ; 9924
                          57 00000000G   00  9E 00002     MOVAB   LIB$SIGNAL, R7
                          56 00000000'   EF  9E 00009     MOVAB   TERMINATOR_CODE, R6
                          53             01  D0 00010     MOVL    #1, I                   ; 9973
                          54       D8    A6  D0 00013 1$: MOVL    EXPRESSION_RADIX, SAVED_RADIX  ; 9982
                    D8    A6             0A  D0 00017     MOVL    #10, EXPRESSION_RADIX   ; 9983
                          00000000'      EF  9F 0001B     PUSHAB  BIT_SELECT_TERM_TBL     ; 9984
                                         7E  D4 00021     CLRL    -(SP)
                  D7D6    CF             02  FB 00023     CALLS   #2, DBG$EXPRESSION_PARSER
                          55             50  D0 00028     MOVL    R0, VALPTR
                    D8    A6             54  D0 0002B     MOVL    SAVED_RADIX, EXPRESSION_RADIX  ; 9985
                                         66  D5 0002F     TSTL    TERMINATOR_CODE         ; 9992
                                         09  12 00031     BNEQ    2$
                          00028E90       8F  DD 00033     PUSHL   #167568
                          67             01  FB 00039     CALLS   #1, LIB$SIGNAL
                  FBCC    C6       04    A6  C0 0003C 2$: ADDL2   TERMINATOR_LENGTH, CHARPTR  ; 9993
                                         55  DD 00042     PUSHL   VALPTR                  ; 9998
                  F65A    CF             01  FB 00044     CALLS   #1, CONVERT_TO_INTEGER
                          52             50  D0 00049     MOVL    R0, VALUE
                          01             53  D1 0004C     CMPL    I, #1                   ; 1000
                                         30  12 0004F     BNEQ    5$
                                         52  D5 00051     TSTL    VALUE                   ; 1000
                                         09  19 00053     BLSS    3$
              00007FFF    8F             52  D1 00055     CMPL    VALUE, #32767
                                         0D  15 0005C     BLEQ    4$
                                         52  DD 0005E 3$: PUSHL   VALUE                   ; 1001
                                         01  DD 00060     PUSHL   #1
```

```
                00028EC8  8F  DD 00062          PUSHL   #167624                                ;
                      67  03  FB 00068          CALLS   #3, LIB$SIGNAL                      ;
                      50        04  AC  D0 0006B 4$:     MOVL    TOKEN, R0                   ; 1001
                08    A0        52  B0 0006F          MOVW    VALUE, 8(R0)                ; 1001
                      0A        66  D1 00073          CMPL    TERMINATOR_CODE, #10        ; 1001
                                09  12 00076          BNEQ    5$                          ; 1001
                00028F00  8F  DD 00078          PUSHL   #167680                             ; 1001
                      67  01  FB 0007E          CALLS   #1, LIB$SIGNAL                      ;
                      02        53  D1 00081 5$:     CMPL    I, #2                       ; 1002
                                23  12 00084          BNEQ    8$                          ;
                                52  D5 00086          TSTL    VALUE                       ; 1002
                                05  19 00088          BLSS    6$                          ;
                      20        52  D1 0008A          CMPL    VALUE, #32                  ;
                                0D  15 0008D          BLEQ    7$                          ;
                                52  DD 0008F 6$:     PUSHL   VALUE                       ; 1002
                                01  DD 00091          PUSHL   #1                          ;
                00028ED0  8F  DD 00093          PUSHL   #167632                             ;
                      67  03  FB 00099          CALLS   #3, LIB$SIGNAL                      ;
                      50        04  AC  D0 0009C 7$:     MOVL    TOKEN, R0                   ; 1003
                0A    A0        52  B0 000A0          MOVW    VALUE, 10(R0)               ; 1003
                      0A        66  D1 000A4          CMPL    TERMINATOR_CODE, #10        ; 1003
                                36  13 000A7          BEQL    11$                         ;
                      03        53  D1 000A9 8$:     CMPL    I, #3                       ; 1004
                                2B  12 000AC          BNEQ    10$                         ;
                                52  D5 000AE          TSTL    VALUE                       ; 1004
                                12  13 000B0          BEQL    9$                          ;
                      01        52  D1 000B2          CMPL    VALUE, #1                   ;
                                0D  13 000B5          BEQL    9$                          ;
                                52  DD 000B7          PUSHL   VALUE                       ; 1004
                                01  DD 000B9          PUSHL   #1                          ;
                00028140  8F  DD 000BB          PUSHL   #164160                             ;
                      67  03  FB 000C1          CALLS   #3, LIB$SIGNAL                      ;
          04    BC        01        0A  52  F0 000C4 9$:     INSV    VALUE, #10, #1, @TOKEN      ; 1004
                                0A  66  D1 000CA          CMPL    TERMINATOR_CODE, #10        ; 1004
                                10  13 000CD          BEQL    11$                         ;
                00028F00  8F  DD 000CF          PUSHL   #167680                             ; 1005
                      67  01  FB 000D5          CALLS   #1, LIB$SIGNAL                      ;
                                04  000D8          RET                                     ; 1004
            FF34        53        01        03  F1  000D9 10$:    ACBL    #3, #1, I, 1$       ; 9973
                                04  000DF 11$:    RET                                     ; 1006

; Routine Size:  224 bytes,    Routine Base:  DBG$CODE + 2DFA
```

```
;  9968     10062  1     ROUTINE GET_RECORD_COMPONENT(PRIMPTR, COMPNAME): NOVALUE =
;  9969     10063  1     !
;  9970     10064  1     ! FUNCTION
;  9971     10065  1     !       This routine is called during the parsing of Primary Symbols to do
;  9972     10066  1     !       record component selection.  It accepts as input a Primary Descriptor
;  9973     10067  1     !       for a record and the name of a record component to be selected from
;  9974     10068  1     !       that record.  It then checks that the Primary Descriptor is indeed
;  9975     10069  1     !       for a record (otherwise component selection is not allowed and an
;  9976     10070  1     !       error is signalled).  It then looks up the component name in the
;  9977     10071  1     !       RST and gets the SYMID for the specified component of the specified
;  9978     10072  1     !       record.  If no such component exists for this record, an error is
;  9979     10073  1     !       signalled.  Finally, this component SYMID is converted to a record
;  9980     10074  1     !       component index which is stored in the Record Sub-Node in the Primary
;  9981     10075  1     !       Descriptor and another Sub-Node is appended for the record component
;  9982     10076  1     !       itself.  The output of the routine is thus the side-effect of modi-
;  9983     10077  1     !       fying the input Primary Descriptor.
;  9984     10078  1     !
;  9985     10079  1     ! INPUTS
;  9986     10080  1     !       PRIMPTR - A pointer to the Primary Descriptor for the record on which
;  9987     10081  1     !                 component selection is to be done.
;  9988     10082  1     !
;  9989     10083  1     !       COMPNAME - A pointer to the name of the record component to be
;  9990     10084  1     !                  selected.  The name must be in Counted ASCII format.
;  9991     10085  1     !
;  9992     10086  1     ! OUTPUTS
;  9993     10087  1     !       The PRIMPTR Primary Descriptor is modified by filling in the record
;  9994     10088  1     !                 component index for the selected component and by appending
;  9995     10089  1     !                 another Primary Descriptor Sub-Node for the component.  The
;  9996     10090  1     !                 PRIMPTR pointer itself is not modified, however.
;  9997     10091  1     !
;  9998     10092  1
;  9999     10093  2         BEGIN
; 10000     10094  2
; 10001     10095  2         MAP
; 10002     10096  2             PRIMPTR: REF DBG$PRIMARY;          ! Pointer to Primary Descriptor
; 10003     10097  2
; 10004     10098  2         LOCAL
; 10005     10099  2             COMP_LIST: REF VECTOR[],           ! List of potential
; 10006     10100  2                                                !      component symids
; 10007     10101  2             COMP_LIST_SIZE,                    ! Length of COMP_LIST
; 10008     10102  2             COMPSYMID: REF RST$ENTRY,          ! SYMID for current record component
; 10009     10103  2             EXACT_MATCH,                       ! Flag saying we've found the record component
; 10010     10104  2             FCODE,                             ! The FCODE of the record component
; 10011     10105  2             NODEPTR: REF DBG$PRIM_NODE,        ! Pointer to Record Sub-Node in the
; 10012     10106  2                                                !      Primary Descriptor
; 10013     10107  2             STATUS,                            ! Status returned by GET_RECORD_VARIANT
; 10014     10108  2             SYMID: REF RST$ENTRY,              ! The SYMID of the record component
; 10015     10109  2             TYPCOMPLST: REF VECTOR[,LONG],     ! Pointer to list of record component
; 10016     10110  2                                                !      SYMIDs in record Type RST Entry
; 10017     10111  2             TYPEID: REF RST$ENTRY,             ! The Type ID of the record record type
; 10018     10112  2                                                !      or of the record component
; 10019     10113  2             VARPTR: REF RST$VAR_ENTRY,         ! Pointer to current RST Variant Entry
; 10020     10114  2             VARSETPTR: REF RST$ENTRY;          ! Pointer to Variant Set RST Entry
; 10021     10115  2
; 10022     10116  2         DBG$GL_CURRENT_PRIMARY = .PRIMPTR;
; 10023     10117  2
; 10024     10118  2
```

```
;10025     10119  2          ! Check that the Primary Descriptor is for a record--otherwise record
;10026     10120  2          ! component selection is not allowed.
;10027     10121  2          !
;10028     10122  2          NODEPTR = .PRIMPTR[DBG$L_PRIM_BLINK];
;10029     10123  2          IF .ENFORCE_RECORD
;10030     10124  2          THEN
;10031     10125  3              BEGIN
;10032     10126  3
;10033     10127  3
;10034     10128  3              ! For ADA, the pointer dereference is implicit. That is, if
;10035     10129  3              ! PTR is a pointer then PTR.C is equivalent to PASCAL's
;10036     10130  3              ! PTR^.C
;10037     10131  3              !
;10038     10132  3              IF .DBG$GB_LANGUAGE EQL DBG$K_ADA
;10039     10133  3              THEN
;10040     10134  4                  BEGIN
;10041     10135  4
;10042     10136  4                  ! First check for the special case ".ALL", which
;10043     10137  4                  ! means pointer dereference.
;10044     10138  4                  !
;10045     10139  4                  IF CH$EQL(4, .COMPNAME,
;10046     10140  4                            4, UPLIT BYTE (3, 'A', 'L', 'L'))
;10047     10141  4                  THEN
;10048     10142  5                      BEGIN
;10049     10143  5                      GET_DEREFERENCE(.PRIMPTR);
;10050     10144  5                      RETURN;
;10051     10145  4                      END;
;10052     10146  4
;10053     10147  4                  ! Now do implicit dereference of the pointer.
;10054     10148  4                  !
;10055     10149  4                  WHILE .PRIMPTR[DBG$B_DHDR_FCODE] EQL RST$K_TYPE_TPTR DO
;10056     10150  4                      GET_DEREFERENCE(.PRIMPTR);
;10057     10151  4                  NODEPTR = .PRIMPTR[DBG$L_PRIM_BLINK];
;10058     10152  3                  END;
;10059     10153  3
;10060     10154  3              IF .PRIMPTR[DBG$B_DHDR_FCODE] NEQ RST$K_TYPE_RECORD
;10061     10155  3              THEN
;10062     10156  3                  SIGNAL(DBG$_NOTRECORD, 1, .COMPNAME);
;10063     10157  3
;10064     10158  3              IF .NODEPTR[DBG$B_PNODE_FCODE] NEQ RST$K_TYPE_RECORD
;10065     10159  3              THEN
;10066     10160  3                  $DBG_ERROR('DBGPARSER\GET_RECORD_COMPONENT 10');
;10067     10161  3              END
;10068     10162  2
;10069     10163  2
;10070     10164  2          ! For languages which allow non-records to be component-selected
;10071     10165  2          ! (i.e., language C), we do not do the above check. In fact,
;10072     10166  2          ! we explicitly change the FCODE to say "record" so that
;10073     10167  2          ! the Primary is processed as a record by routines in DBGVALUES
;10074     10168  2          ! and in DBGPRINT.
;10075     10169  2          !
;10076     10170  2          ELSE
;10077     10171  2              NODEPTR[DBG$B_PNODE_FCODE] = RST$K_TYPE_RECORD;
;10078     10172  2
;10079     10173  2
;10080     10174  2          ! Search the RST Hash Table for all record components of this name.
;10081     10175  2          ! If we find one which belongs to the given record, then light the
```

```
:10082          10176  2          ! EXACT_MATCH flag and exit the loop. If we find one but it belongs
:10083          10177  2          ! to the wrong record, we save its SYMID anyway - some languages
:10084          10178  2          ! allow this. If we find more than one but they all belong to the
:10085          10179  2          ! wrong record, then light DUPLICATE_FLAG - this will be an error.
:10086          10180  2          !
:10087          10181  2          TYF_D = .NODEPTR[DBG$L_PNODE_TYPEID];
:10088          10182  2          SYMID = 0;
:10089          10183  2          EXACT_MATCH = FALSE;
:10090          10184  2          COMP_LIST_SIZE = 10;
:10091          10185  2          COMP_LIST = DBG$GET_TEMPMEM(.COMP_LIST_SIZE);
:10092          10186  2          COMP_LIST[0] = 0;
:10093          10187  2          DBG$HASH_FIND_SETUP(.COMPNAME);
:10094          10188  2          WHILE TRUE DO
:10095          10189  3              BEGIN
:10096          10190  3              SYMID = DBG$HASH_FIND(.COMPNAME);
:10097          10191  3              IF .SYMID EQL 0 THEN EXITLOOP;
:10098          10192  3              IF .SYMID[RST$B_KIND] EQL RST$K_TYPCOMP
:10099          10193  3              THEN
:10100          10194  4                  BEGIN
:10101          10195  4                  IF .SYMID[RST$L_UPSCOPEPTR] EQL .TYPEID
:10102          10196  4                  THEN
:10103          10197  5                      BEGIN
:10104          10198  5
:10105          10199  5
:10106          10200  5                      ! This is the case where we find a component with the right
:10107          10201  5                      ! name belonging to the right record.
:10108          10202  5                      !
:10109          10203  5                      EXACT_MATCH = TRUE;
:10110          10204  5                      EXITLOOP;
:10111          10205  5                      END
:10112          10206  5
:10113          10207  4                  ELSE
:10114          10208  5                      BEGIN
:10115          10209  5                      COMP_LIST[0] = .COMP_LIST[0] + 1;
:10116          10210  5
:10117          10211  5                      ! If we overflow the component list then expand it.
:10118          10212  5                      !
:10119          10213  5                      IF .COMP_LIST[0] GEQ .COMP_LIST_SIZE
:10120          10214  5                      THEN
:10121          10215  6                          BEGIN
:10122          10216  6                          LOCAL
:10123          10217  6                              SAVE_COMP_LIST;
:10124          10218  6                          SAVE_COMP_LIST = .COMP_LIST;
:10125          10219  6                          COMP_LIST = DBG$GET_TEMPMEM(.COMP_LIST_SIZE+10);
:10126          10220  6                          CH$MOVE(4*.COMP_LIST_SIZE, .SAVE_COMP_LIST, .COMP_LIST);
:10127          10221  6                          COMP_LIST_SIZE = .COMP_LIST_SIZE + 10;
:10128          10222  5                          END;
:10129          10223  5                      COMP_LIST[.COMP_LIST[0]] = .SYMID;
:10130          10224  4                      END;
:10131          10225  3                  END;
:10132          10226  2              END;
:10133          10227  2
:10134          10228  2
:10135          10229  2          ! Signal errors for the cases where a record component was not found,
:10136          10230  2          ! or where the reference is ambiguous.
:10137          10231  2          !
:10138          10232  2          IF NOT .EXACT_MATCH
```

```
;10139      10233  2          THEN
;10140      10234  2              IF .ENFORCE_RECORD AND NOT .INCOMPLETE_QUAL
;10141      10235  2              THEN
;10142      10236  2
;10143      10237  2                  ! We did not find a component in the right record, and this
;10144      10238  2                  ! language is strict about membership checking. So signal
;10145      10239  2                  ! an error.
;10146      10240  2                  !
;10147      10241  2                  SIGNAL(DBG$_NOFIELD, 1, .COMPNAME)
;10148      10242  2
;10149      10243  2              ELSE
;10150      10244  2                  IF .COMP_LIST[0] EQL 0
;10151      10245  2                  THEN
;10152      10246  2
;10153      10247  2
;10154      10248  2                      ! We did not find any components at all. Signal an error.
;10155      10249  2                      !
;10156      10250  2                      SIGNAL(DBG$_NOFIELD, 1, .COMPNAME)
;10157      10251  2
;10158      10252  2                  ELSE
;10159      10253  2
;10160      10254  2
;10161      10255  2                      ! Call a routine which attempts to resolve amiguities.
;10162      10256  2                      ! If it fails, it will return false and we signal an
;10163      10257  2                      ! error saying that the record reference was ambiguous.
;10164      10258  2                      !
;10165      10259  2                      IF NOT RESOLVE_COMPONENT(.TYPEID, .COMP_LIST, SYMID,
;10166      10260  2                                               .PRIMPTR, .COMPNAME)
;10167      10261  2                      THEN
;10168      10262  2                          SIGNAL(DBG$_AMBFIELD, 1, .COMPNAME);
;10169      10263  2
;10170      10264  2
;10171      10265  2          ! Set the EVAL bit in the Record Sub-Node to indicate that a record com-
;10172      10266  2          ! ponent has actually been selected.
;10173      10267  2          !
;10174      10268  2          NODEPTR = .PRIMPTR[DBG$L_PRIM_BLINK];
;10175      10269  2          NODEPTR[DBG$V_PNODE_EVAL] = TRUE;
;10176      10270  2
;10177      10271  2
;10178      10272  2          ! We have found a Type Component SYMID for a component of the current
;10179      10273  2          ! record data type.  Now convert that SYMID to a record component index
;10180      10274  2          ! into the component vector for the record type.  We do this by searching
;10181      10275  2          ! the component list in the record type's Type RST Entry.
;10182      10276  2          !
;10183      10277  2          TYPCOMPLST = TYPEID[RST$A_TYPCOMPLST];
;10184      10278  2          INCR I FROM 0 TO .TYPEID[RST$L_TYPCOMPCNT] - 1 DO
;10185      10279  3              BEGIN
;10186      10280  3              COMPSYMID = .TYPCOMPLST[.I];
;10187      10281  3
;10188      10282  3
;10189      10283  3                  ! If this component is the one we seek, set its index into the Record
;10190      10284  3                  ! Sub-Node and leave the loop.
;10191      10285  3                  !
;10192      10286  3                  IF .SYMID EQL .COMPSYMID
;10193      10287  3                  THEN
;10194      10288  4                      BEGIN
;10195      10289  4                      NODEPTR[DBG$W_PNREC_INDEX] = .I + 1;
```

```
;10196      10290  4            EXITLOOP;
;10197      10291  3            END;
;10198      10292  3
;10199      10293  3
;10200      10294  3        ! If this record component is a Variant Set, see if the desired
;10201      10295  3        ! component is part of one of the variants in this Variant Set.
;10202      10296  3        !
;10203      10297  3        IF .COMPSYMID[RST$B_KIND] EQL RST$K_VARIANT
;10204      10298  3        THEN
;10205      10299  4            BEGIN
;10206      10300  4            VARSTK_INDEX = 0;
;10207      10301  4            STATUS = GET_RECORD_VARIANT(.COMPSYMID, .SYMID);
;10208      10302  4            IF .STATUS
;10209      10303  4            THEN
;10210      10304  5                BEGIN
;10211      10305  5
;10212      10306  5
;10213      10307  5                ! We found the record component in the current Variant Set.
;10214      10308  5                ! Set the index of the Variant Set in the Record Sub-Node.
;10215      10309  5                ! Then build all necessary Primary Descriptor Variant Sub-
;10216      10310  5                ! Nodes, one for each level of variant nesting.
;10217      10311  5                !
;10218      10312  5                NODEPTR[DBG$W_PNREC_INDEX] = .I + 1;
;10219      10313  5                INCR J FROM 0 TO .VARSTK_INDEX - 1 DO
;10220      10314  6                    BEGIN
;10221      10315  6                    DBG$BUILD_PRIMARY_SUBNODE(.PRIMPTR, RST$K_VARIANT,
;10222      10316  6                                              0, RST$K_TYPE_VARIANT, 0, 0);
;10223      10317  6                    VARSETPTR = .VARSTACK1[.J];
;10224      10318  6                    VARPTR = .VARSTACK2[.J];
;10225      10319  6                    NODEPTR = .PRIMPTR[DBG$L_PRIM_BLINK];
;10226      10320  6                    NODEPTR[DBG$V_PNODE_EVAL]    = TRUE;
;10227      10321  6                    NODEPTR[DBG$L_PNVAR_TAGID]   = .VARSETPTR[RST$L_VARTAGPTR];
;10228      10322  6                    NODEPTR[DBG$W_PNVAR_INDEX]   = .VARSTACK3[.J];
;10229      10323  6                    NODEPTR[DBG$W_PNVAR_NCOMPS]  = .VARPTR[RST$L_VAR_COMPCNT];
;10230      10324  6                    NODEPTR[DBG$L_PNVAR_COMPLST] =  VARPTR[RST$A_VAR_COMPLST];
;10231      10325  6                    NODEPTR[DBG$L_PNVAR_DSTPTR]  = .VARPTR[RST$L_VAR_DSTPTR];
;10232      10326  5                    END;
;10233      10327  5
;10234      10328  5
;10235      10329  5                ! The Variant Sub-Nodes have successfully been constructed.
;10236      10330  5                ! Now exit the search of the record component list so that
;10237      10331  5                ! we can build the Sub-Node for the component actually found.
;10238      10332  5                !
;10239      10333  5                EXITLOOP;
;10240      10334  4                END;
;10241      10335  4
;10242      10336  3            END;                        ! End of variant code
;10243      10337  3
;10244      10338  2        END;                            ! End of loop over record components
;10245      10339  2
;10246      10340  2
;10247      10341  2        ! Finally append another Primary Descriptor Sub-Node for the selected
;10248      10342  2        ! record component.  Then return.
;10249      10343  2        !
;10250      10344  2        DBG$STA_SYMTYPE(.SYMID, FCODE, TYPEID);
;10251      10345  2        DBG$BUILD_PRIMARY_SUBNODE(.PRIMPTR,
;10252      10346  2                                  RST$K_TYPCOMP, .SYMID, .FCODE, .TYPEID, 0);
```

```
;10253          10347  2          RETURN;
;10254          10348  2
;10255          10349  1          END;


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

                                                  03  03372 P.AYJ:  .BYTE   3                                        ;
                                                  41  03373        .ASCII  \A\                                       ;
                                                  4C  03374        .ASCII  \L\                                       ;
                                                  4C  03375        .ASCII  \L\                                       ;
5F  54  45  47  5C  52  45  53  52  41  50  47  42  44  21  03376 P.AYK:  .ASCII  \!DBGPARSER\<92>\GET_RECORD_COMPONENT 10\  ;
4E  45  4E  4F  50  4D  4F  43  5F  44  52  4F  43  45  52  03385                                                        ;
                                        30  31  20  54  03394                                                           ;


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

                                              0FFC 00000 GET_RECORD_COMPONENT:
                                                        .WORD   Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11          ; 1006
                                 5E             10  C2 00002        SUBL2   #16, SP                                 ;
                                 5A       04    AC  D0 00005        MOVL    PRIMPTR, R10                            ; 1011
                00000000G        00             5A  D0 00009        MOVL    R10, DBG$GL_CURRENT_PRIMARY             ;
                                 57       18    AA  D0 00010        MOVL    24(R10), NODEPTR                        ; 1012
                                 63 00000000'   EF  E9 00014        BLBC    ENFORCE_RECORD, 5$                      ; 1012
                                 09 00000000G   00  91 0001B        CMPB    DBG$GB_LANGUAGE, #9                     ; 1013
                                                25  12 00022        BNEQ    3$                                      ;
                00000000'        EF       08    BC  D1 00024        CMPL    @COMPNAME, P.AYJ                        ; 1013
                                                08  12 0002C        BNEQ    1$                                      ;
                                                5A  DD 0002E        PUSHL   R10                                     ; 1014
                    FE78         CF             01  FB 00030        CALLS   #1, GET_DEREFERENCE                     ;
                                                04  00035        RET                                             ; 1014
                                 06       06    AA  91 00036 1$:    CMPB    6(R10), #6                              ; 1014
                                                09  12 0003A        BNEQ    2$                                      ;
                                                5A  DD 0003C        PUSHL   R10                                     ; 1015
                    FE6A         CF             01  FB 0003E        CALLS   #1, GET_DEREFERENCE                     ;
                                                F1  11 00043        BRB     1$                                      ;
                                 57       18    AA  D0 00045 2$:    MOVL    24(R10), NODEPTR                        ; 1015
                                 07       06    AA  91 00049 3$:    CMPB    6(R10), #7                              ; 1015
                                                12  13 0004D        BEQL    4$                                      ;
                                          08    AC  DD 0004F        PUSHL   COMPNAME                                ; 1015
                                                01  DD 00052        PUSHL   #1                                      ;
                                    00028E98    8F  DD 00054        PUSHL   #167576                                 ;
                00000000G        00             03  FB 0005A        CALLS   #3, LIB$SIGNAL                          ;
                                 07       09    A7  91 00061 4$:    CMPB    9(NODEPTR), #7                          ; 1015
                                                1B  13 00065        BEQL    6$                                      ;
                                    00000000'   EF  9F 00067        PUSHAB  P.AYK                                   ; 1016
                                                01  DD 0006D        PUSHL   #1                                      ;
                                    00028362    8F  DD 0006F        PUSHL   #164706                                 ;
                00000000G        00             03  FB 00075        CALLS   #3, LIB$SIGNAL                          ;
                                                04  11 0007C        BRB     6$                                      ; 1012
                    09           A7             07  90 0007E 5$:    MOVB    #7, 9(NODEPTR)                          ; 1017
                    08           AE       0C    A7  D0 00082 6$:    MOVL    12(NODEPTR), TYPEID                     ; 1018
                                                6E  7C 00087        CLRQ    EXACT_MATCH                             ; 1018
                                 56             0A  D0 00089        MOVL    #10, COMP_LIST_SIZE                     ; 1018
```

```
                                56  DD 0008C          PUSHL   COMP_LIST_SIZE                          ; 1018
                00000000G   00      01  FB 0008E          CALLS   #1, DBG$GET_TEMPMEM
                            59      50  D0 00095          MOVL    R0, COMP_LIST
                                    69  D4 00098          CLRL    (COMP_LIST)                         ; 1018
                            5B  08  AC  D0 0009A          MOVL    COMPNAME, R11                       ; 1018
                                5B  DD 0009E          PUSHL   R11
                00C00000G   00      01  FB 000A0          CALLS   #1, DBG$HASH_FIND_SETUP
                                5B  DD 000A7 7$:      PUSHL   R11                                 ; 1019
                00000000G   00      01  FB 000A9          CALLS   #1, DBG$HASH_FIND
                      04    AE      50  D0 000B0          MOVL    R0, SYMID
                            58  04  AE  D0 000B4          MOVL    SYMID, R8                           ; 1019
                                    3D  13 000B8          BEQL    10$
                            0A  14  A8  91 000BA          CMPB    20(R8), #10                         ; 1019
                                    E7  12 000BE          BNEQ    7$
                      08    AE  10  A8  D1 000C0          CMPL    16(R8), TYPEID                      ; 1019
                                    05  12 000C5          BNEQ    8$
                            6E      01  D0 000C7          MOVL    #1, EXACT_MATCH                     ; 1020
                                    2B  11 000CA          BRB     10$                                 ; 1019
                                    69  D6 000CC 8$:       INCL    (COMP_LIST)                         ; 1020
                            56      69  D1 000CE          CMPL    (COMP_LIST), COMP_LIST_SIZE         ; 1021
                                    1B  19 000D1          BLSS    9$
                            52      59  D0 000D3          MOVL    COMP_LIST, SAVE_COMP_LIST           ; 1021
                                0A  A6  9F 000D6          PUSHAB  10(COMP_LIST_SIZE)                  ; 1021
                00000000G   00      01  FB 000D9          CALLS   #1, DBG$GET_TEMPMEM
                            59      50  D0 000E0          MOVL    R0, COMP_LIST
                50          56      02  78 000E3          ASHL    #2, COMP_LIST_SIZE, R0              ; 1022
                69          62      50  28 000E7          MOVC3   R0, (SAVE_COMP_LIST), (COMP_LIST)
                            56      0A  C0 000EB          ADDL2   #10, COMP_LIST_SIZE                 ; 1022
                            50      69  D0 000EE 9$:       MOVL    (COMP_LIST), R0                     ; 1022
                          6940      58  D0 000F1          MOVL    R8, (COMP_LIST)[R0]
                                    B0  11 000F5          BRB     7$                                  ; 1018
                            42      6E  E8 000F7 10$:     BLBS    EXACT_MATCH, 15$                    ; 1023
                            07  00000000'  EF  E9 000FA          BLBC    ENFORCE_RECORD, 11$         ; 1023
                            04  00000000'  EF  E9 00101          BLBC    INCOMPLETE_QUAL, 12$
                                    69  D5 00108 11$:      TSTL    (COMP_LIST)                         ; 1024
                                    0C  12 0010A          BNEQ    13$
                                5B  DD 0010C 12$:      PUSHL   R11                                 ; 1025
                                01  DD 0010E          PUSHL   #1
                            00028C80    8F  DD 00110          PUSHL   #167040
                                    1D  11 00116          BRB     14$
                            7E          5A  7D 00118 13$:      MOVQ    R10, -(SP)                  ; 1026
                                0C  AE  9F 0011B          PUSHAB  SYMID                               ; 1025
                                59  DD 0011E          PUSHL   COMP_LIST
                                18  AE  DD 00120          PUSHL   TYPEID
                      0000V CF      05  FB 00123          CALLS   #5, RESOLVE_COMPONENT
                            11      50  E8 00128          BLBS    R0, 15$
                                5B  DD 0012B          PUSHL   R11                                 ; 1026
                                01  DD 0012D          PUSHL   #1
                            00028F58    8F  DD 0012F          PUSHL   #167768
                00000000G   00      03  FB 00135 14$:      CALLS   #3, LIB$SIGNAL
                            57      18  AA  D0 0013C 15$:      MOVL    24(R10), NODEPTR            ; 1026
                      0A    A7      01  88 00140          BISB2   #1, 10(NODEPTR)                     ; 1026
                            58  08  AE  D0 00144          MOVL    TYPEID, R8                          ; 1027
                            52  2C  A8  9E 00148          MOVAB   44(R8), TYPCOMPLST
                            54      01  CE 0014C          MNEGL   #1, I                               ; 1028
                                    0086  31 0014F          BRW     20$
                            56          6244  D0 00152 16$:      MOVL    (TYPCOMPLST)[I], COMPSYMID  ; 1028
```

```
                         56        04    AE  D1 00156          CMPL    SYMID, COMPSYMID                        ; 1028
                                         07  12 0015A          BNEQ    17$                                     ;
                18   A7                 54  01 A1 0015C       ADDW3   #1, I, 24(NODEPTR)                      ; 1028
                                         7F  11 00161          BRB     22$                                     ; 1028
                         0B        14    A6  91 00163 17$:     CMPB    20(COMPSYMID), #11                      ; 1029
                                         6F  12 00167          BNEQ    20$                                     ;
                            00000000'    EF  D4 00169          CLRL    VARSTK_INDEX                            ; 1030
                                   04    AE  DD 0016F          PUSHL   SYMID                                   ; 1030
                                         56  DD 00172          PUSHL   COMPSYMID                               ;
                      0000V  CF          02  FB 00174          CALLS   #2, GET_RECORD_VARIANT                  ;
                         6E              50  D0 00179          MOVL    R0, STATUS                              ;
                         59              6E  E9 0017C          BLBC    STATUS, 20$                             ; 1030
                18   A7                 54  01 A1 0017F       ADDW3   #1, I, 24(NODEPTR)                      ; 1031
                         5B 00000000'    EF  D0 00184          MOVL    VARSTK_INDEX, R11                       ; 1031
                         53              01  CE 0018B          MNEGL   #1, J                                   ;
                                         42  11 0018E          BRB     19$                                     ;
                                         7E  7C 00190 18$:     CLRQ    -(SP)                                   ; 1031
                                         13  DD 00192          PUSHL   #19                                     ;
                         7E              0B  7D 00194          MOVQ    #11, -(SP)                              ;
                                         5A  DD 00197          PUSHL   R10                                     ;
                      CFF4   CF          06  FB 00199          CALLS   #6, DBG$BUILD_PRIMARY_SUBNODE           ;
                         59 00000000'EF  43  D0 0019E          MOVL    VARSTACK1[J], VARSETPTR                 ; 1031
                         55 00000000'EF  43  D0 001A6          MOVL    VARSTACK2[J], VARPTR                    ; 1031
                         57        18    AA  D0 001AE          MOVL    24(R10), NODEPTR                        ; 1031
                      0A   A7           01  88 001B2          BISB2   #1, 10(NODEPTR)                         ; 1032
                      1C   A7     10    A9  D0 001B6          MOVL    16(VARSETPTR), 28(NODEPTR)              ; 1032
                      18   A7 00000000'EF  43  F7 001BB          CVTLW   VARSTACK3[J], 24(NODEPTR)               ; 1032
                      1A   A7     04    A5  B0 001C4          MOVW    4(VARPTR), 26(NODEPTR)                  ; 1032
                      20   A7     08    A5  9E 001C9          MOVAB   8(R5), 32(NODEPTR)                      ; 1032
                      24   A7           65  D0 001CE          MOVL    (VARPTR), 36(NODEPTR)                   ; 1032
                   BA              53        5B  F2 001D2 19$:     AOBLSS  R11, J, 18$                             ; 1031
                                             0A  11 001D6          BRB     22$                                     ; 1030
                   02              54  28    A8  F2 001D8 20$:     AOBLSS  40(R8), I, 21$                          ; 1027
                                             03  11 001DD          BRB     22$                                     ;
                                       FF70  31  001DF 21$:        BRW     16$                                     ;
                                         08  AE  9F 001E2 22$:     PUSHAB  TYPEID                                  ; 1034
                                         10  AE  9F 001E5         PUSHAB  FCODE                                   ;
                                         0C  AE  DD 001E8         PUSHL   SYMID                                   ;
                         00000000G       00  03  FB 001EB         CALLS   #3, DBG$STA_SYMTYPE                     ;
                                             7E  D4 001F2         CLRL    -(SP)                                   ; 1034
                                         0C  AE  DD 001F4         PUSHL   TYPEID                                  ; 1034
                                         14  AE  DD 001F7         PUSHL   FCODE                                   ;
                                         10  AE  DD 001FA         PUSHL   SYMID                                   ;
                                             0A  DD 001FD         PUSHL   #10                                     ; 1034
                                             5A  DD 001FF         PUSHL   R10                                     ;
                              CF8C   CF          06  FB 00201         CALLS   #6, DBG$BUILD_PRIMARY_SUBNODE           ;
                                                 04  00206         RET                                             ; 1034

; Routine Size:  519 bytes,    Routine Base:  DBG$CODE + 2EDA
```

```
;10257      10350  1     ROUTINE GET_RECORD_VARIANT(VARSETPTR, SYMID) =
;10258      10351  1     !
;10259      10352  1     ! FUNCTION
;10260      10353  1     !       This routine looks for a record component with a known SYMID among all
;10261      10354  1     !       the variants in a specified Variant Set.  It returns TRUE if the compo-
;10262      10355  1     !       nent is found in that Variant Set, and as a side-effect, it also builds
;10263      10356  1     !       a "Variant Stack" which specifies which sequence of variants and Vari-
;10264      10357  1     !       ant Sets contain the component.  This routine calls itself recursively
;10265      10358  1     !       to search variants within variants; the "Variant Stack" records the
;10266      10359  1     !       path taken through the tree of variants to reach the desired component.
;10267      10360  1     !       This stack is then used by GET_RECORD_COMPONENT to build the Primary
;10268      10361  1     !       Descriptor Variant Sub-Nodes needed to describe that path.
;10269      10362  1     !
;10270      10363  1     ! INPUTS
;10271      10364  1     !       VARSETPTR - Pointer to the Variant Set to be searched for the SYMID
;10272      10365  1     !                   record component.
;10273      10366  1     !
;10274      10367  1     !       SYMID     - The SYMID (RST Entry address) of the record component to
;10275      10368  1     !                   search for among the variants of the current Variant Set.
;10276      10369  1     !                   (This SYMID has been found by looking up the record compo-
;10277      10370  1     !                   nent by its name.)
;10278      10371  1     !
;10279      10372  1     ! OUTPUTS
;10280      10373  1     !       If the SYMID record component is found in any of the VARSETPTR variants,
;10281      10374  1     !                   this routine returns TRUE as its value.  If the SYMID compo-
;10282      10375  1     !                   nent is not found, it returns FALSE.  If TRUE is returned,
;10283      10376  1     !                   the Variant Stack (in OWN storage) contains the list of Vari-
;10284      10377  1     !                   ant Sets and variants which contain the SYMID component.
;10285      10378  1     !
;10286      10379  1
;10287      10380  2         BEGIN
;10288      10381  2
;10289      10382  2         MAP
;10290      10383  2             VARSETPTR: REF RST$ENTRY;           ! Pointer to Variant Set RST Entry
;10291      10384  2
;10292      10385  2         LOCAL
;10293      10386  2             COMPLST: REF VECTOR[,LONG],         ! Pointer to vector of component RST
;10294      10387  2                                                 !     pointers in RST Variant Entry
;10295      10388  2             COMPPTR: REF RST$ENTRY,             ! Pointer to current variant component's
;10296      10389  2                                                 !     RST entry
;10297      10390  2             VARPTR: REF RST$VAR_ENTRY,          ! Pointer to current RST Variant Entry
;10298      10391  2             VARSETTBL: REF VECTOR[,LONG],       ! Pointer to vector of variant in the
;10299      10392  2                                                 !     Variant Set RST Entry
;10300      10393  2             STATUS;                             ! Status returned by recursive call
;10301      10394  2
;10302      10395  2
;10303      10396  2
;10304      10397  2         ! Push the address of the current Variant Set RST Entry on the Variant
;10305      10398  2         ! Stack maintained by this routine and GET_RECORD_COMPONENT.  This is
;10306      10399  2         ! how we keep track of nested Variant Sets in the record.
;10307      10400  2         !
;10308      10401  2         IF .VARSTK_INDEX GEQ VARSTK_SIZE THEN SIGNAL(DBG$_VARNESDEP);
;10309      10402  2         VARSTK_INDEX = .VARSTK_INDEX + 1;
;10310      10403  2         VARSTACK1[.VARSTK_INDEX - 1] = .VARSETPTR;
;10311      10404  2
;10312      10405  2
;10313      10406  2         ! Loop through all the variants in this Variant Set.  For each variant in
```

```
;10314      10407  2          ; the set, search its components until we find the SYMID component, i.e
;10315      10408  2          ; the record component we are looking for.
;10316      10409  2          !
;10317      10410  2          VARSETTBL = VARSETPTR[RST$A_VARSETTBL];
;10318      10411  2          INCR I FROM 0 TO .VARSETPTR[RST$L_VARSETCNT] - 1 DO
;10319      10412  3              BEGIN
;10320      10413  3              VARPTR = .VARSETTBL[.I];
;10321      10414  3              VARSTACK2[.VARSTK_INDEX - 1] = .VARPTR;
;10322      10415  3
;10323      10416  3
;10324      10417  3              ! Loop over the record components in this particular variant.  If the
;10325      10418  3              ! SYMID component is found, we return TRUE and leave the Variant Stack
;10326      10419  3              ! with its current contents.  If we find another Variant Set component
;10327      10420  3              ! within this variant (i.e., nested variants within the record), this
;10328      10421  3              ! routine calls itself recursively to look for the SYMID component in
;10329      10422  3              ! that nested variant.  If it is found there, TRUE is returned.
;10330      10423  3              !
;10331      10424  3              COMPLST = VARPTR[RST$A_VAR_COMPLST];
;10332      10425  3              INCR J FROM 0 TO .VARPTR[RST$L_VAR_COMPCNT] - 1 DO
;10333      10426  4                  BEGIN
;10334      10427  4                  COMPPTR = .COMPLST[.J];
;10335      10428  4                  VARSTACK3[.VARSTK_INDEX - 1] = .J + 1;
;10336      10429  4                  IF .COMPPTR EQL .SYMID THEN RETURN TRUE;
;10337      10430  4                  IF .COMPPTR[RST$B_KIND] EQL RST$K_VARIANT
;10338      10431  4                  THEN
;10339      10432  5                      BEGIN
;10340      10433  5                      STATUS = GET_RECORD_VARIANT(.COMPPTR, .SYMID);
;10341      10434  5                      IF .STATUS THEN RETURN TRUE;
;10342      10435  4                      END;
;10343      10436  4
;10344      10437  3                  END;                        ! End of loop over variant components
;10345      10438  3
;10346      10439  2              END;                            ! End of loop over Variant Set
;10347      10440  2
;10348      10441  2              ! We did not find the SYMID component anywhere among the variants in this
;10349      10442  2              ! Variant Set.  We thus pop the Variant Stack and return FALSE.
;10350      10443  2              !
;10351      10444  2              VARSTK_INDEX = .VARSTK_INDEX - 1;
;10352      10445  2              RETURN FALSE;
;10353      10446  2
;10354      10447  2              END;
;10355      10448  1


                                03FC 00000 GET_RECORD_VARIANT:
                                           .WORD   Save R2,R3,R4,R5,R6,R7,R8,R9                ; 1035
                    59 00000000'  EF  9E 00002     MOVAB   VARSTK_INDEX, R9
                    14            69  D1 00009     CMPL    VARSTK_INDEX, #20                   ; 1040
                                  0D  19 0000C     BLSS    1$
                       00028A32   8F  DD 0000E     PUSHL   #166450
                00000000G   00    01  FB 00014     CALLS   #1, LIB$SIGNAL
                                  69  D6 0001B 1$:  INCL    VARSTK_INDEX                        ; 1040
                    50            69  D0 0001D     MOVL    VARSTK_INDEX, R0                    ; 1040
                    57       04   AC  D0 00020     MOVL    VARSETPTR, R7
```

```
                FF0C C940          57  D0 00024          MOVL    R7, VARSTACK1-4[R0]                     ;
                      53    18    A7  9E 0002A          MOVAB   24(R7), VARSETTBL                       ; 1041
                      56          01  CE 0002E          MNEGL   #1, I                                   ; 1042
                                  44  11 00031          BRB     6$                                      ;
                      52        6346  D0 00033 2$:      MOVL    (VARSETTBL)[I], VARPTR                  ; 1041
                      51          69  D0 00037          MOVL    VARSTK_INDEX, R1                        ; 1041
                FF5C C941          52  D0 0003A          MOVL    VARPTR, VARSTACK2-4[R1]                 ;
                      54    08    A2  9E 00040          MOVAB   8(R2), COMPLST                          ; 1042
                      55          01  CE 00044          MNEGL   #1, J                                   ; 1042
                                  29  11 00047          BRB     5$                                      ;
                      58        6445  D0 00049 3$:      MOVL    (COMPLST)[J], COMPPTR                   ;
                      51          69  D0 0004D          MOVL    VARSTK_INDEX, R1                        ; 1042
                  AC A941    01    A5  9E 00050          MOVAB   1(R5), VARSTACK3-4[R1]                  ;
                  08    AC          58  D1 00056          CMPL    COMPPTR, SYMID                          ; 1042
                                    12  13 0005A          BEQL    4$                                      ;
                      0B    14    A8  91 0005C          CMPB    20(COMPPTR), #11                        ; 1043
                                    10  12 00060          BNEQ    5$                                      ;
                            08    AC  DD 00062          PUSHL   SYMID                                   ; 1043
                                  58  DD 00065          PUSHL   COMPPTR                                 ;
                    95    AF          02  FB 00067          CALLS   #2, GET_RECORD_VARIANT                  ;
                          04          50  E9 0006B          BLBC    STATUS, 5$                          ; 1043
                          50          01  D0 0006E 4$:      MOVL    #1, R0                              ;
                                      04  00071          RET                                             ;
                D2        55    04    A2  F2 00072 5$:      AOBLSS  4(VARPTR), J, 3$                    ; 1042
                B7        56    08    A7  F2 00077 6$:      AOBLSS  8(R7), I, 2$                        ; 1041
                                  69  D7 0007C          DECL    VARSTK_INDEX                            ; 1044
                                  50  D4 0007E          CLRL    R0                                      ; 1044
                                      04  00080          RET                                             ; 1044

; Routine Size:  129 bytes,    Routine Base:  DBG$CODE + 30E1
```

```
;10357      10449  1        ROUTINE GET_SET_CONSTANT =
;10358      10450  1        !
;10359      10451  1        ! FUNCTION
;10360      10452  1        !       This routine picks up set constants.  It calls DBG$EXPRESSION_PARSER
;10361      10453  1        !       to parse and evaluate each set constants expression.  It also checks
;10362      10454  1        !       the data type of each set constants and converts it to the appropriate
;10363      10455  1        !       data type of the first set constant as necessary.
;10364      10456  1        !
;10365      10457  1        !       This routine assumes that the opening set parenthesis has
;10366      10458  1        !       already been found and that the parse pointer points to the start
;10367      10459  1        !       of the first set constant expression.  When this routine returns,
;10368      10460  1        !       the parse pointer is left pointing at the first character after
;10369      10461  1        !       the closing set parenthesis.
;10370      10462  1        !
;10371      10463  1        ! INPUTS
;10372      10464  1        !       None.
;10373      10465  1        !
;10374      10466  1        ! OUTPUTS
;10375      10467  1        !       Pointer to set constant value descriptor.
;10376      10468  1        !
;10377      10469  1
;10378      10470  2            BEGIN
;10379      10471  2
;10380      10472  2            LOCAL
;10381      10473  2                CREATE,                         ! Flag set to true to create set
;10382      10474  2                                                !      constant value descriptor
;10383      10475  2                LOW_RANGE_VAL,                  ! Low value of a subscript range
;10384      10476  2                SAVED_RADIX,                    ! Temporarily saved expression radix
;10385      10477  2                SETVAL: REF BITVECTOR[],        ! A vector of 256 bits
;10386      10478  2                SETVALPTR: REF DBG$VALDESC,     ! Pointer to Value Descriptor for
;10387      10479  2                                                !      declared subscript data type
;10388      10480  2                THIS_SUBSCR_IS_RANGE,           ! Flag set if the current subscript is
;10389      10481  2                                                !      given as a subscript range
;10390      10482  2                TYPEID: REF RST$ENTRY,          ! Pointer to RST entry
;10391      10483  2                VALADDR: REF VECTOR[,LONG],     ! Pointer to integer subscript value
;10392      10484  2                VALPTR: REF DBG$VALDESC;        ! Pointer to subscript Value Descriptor
;10393      10485  2
;10394      10486  2
;10395      10487  2
;10396      10488  2            ! Check for Empty Set [].  Create a value descriptor for it, mark 'FFFF'
;10397      10489  2            ! in class and dtype fields to indicate this is [], so DBG$EVAL_LANG_OPERATOR
;10398      10490  2            ! can play with it.
;10399      10491  2            !
;10400      10492  2            IF .CHARPTR[0] EQL ']'
;10401      10493  2            THEN
;10402      10494  3                BEGIN
;10403      10495  3                SETVALPTR = DBG$MAKE_SKELETON_DESC(DBG$K_VALUE_DESC, 8*4);
;10404      10496  3                SETVALPTR[DBG$B_DHDR_LANG] = %X'FF';
;10405      10497  3                SETVALPTR[DBG$B_DHDR_KIND] = RST$K_DATA;
;10406      10498  3                SETVALPTR[DBG$B_DHDR_FCODE] = RST$K_TYPE_SET;
;10407      10499  3                SETVALPTR[DBG$L_VALUE_POINTER] = SETVALPTR[DBG$A_VALUE_ADDRESS];
;10408      10500  3                SETVALPTR[DBG$B_VALUE_CLASS] = %X'FF';
;10409      10501  3                SETVALPTR[DBG$B_VALUE_DTYPE] = %X'FF';
;10410      10502  3                SETVALPTR[DBG$W_VALUE_LENGTH] = 32;
;10411      10503  3                CHARPTR = .CHARPTR + 1;
;10412      10504  3                RETURN .SETVALPTR;
;10413      10505  2                END;
```

```
;10414          10506  2
;10415          10507  2
;10416          10508  2          ! Loop through the set constant expressions for this set constant.  Each
;10417          10509  2          ! set constant is parsed, evaluated, and converted to the appropriate type
;10418          10510  2          ! of the first set constant (with the type being checked in the process).
;10419          10511  2          ! Note that TERMINATOR_CODE is set within the loop as a side-effect of the
;10420          10512  2          ! call on DBG$EXPRESSION_PARSER.
;10421          10513  2          !
;10422          10514  2          CREATE = TRUE;
;10423          10515  2          THIS_SUBSCR_IS_RANGE = FALSE;
;10424          10516  2          TERMINATOR_CODE = TOKEN$K_TERM_COMMA;
;10425          10517  2          WHILE .TERMINATOR_CODE NEQ TOKEN$K_TERM_CLOSE DO
;10426          10518  2              BEGIN
;10427          10519  2
;10428          10520  2
;10429          10521  2              ! Call the expression parser to pick up the next set constant expression
;10430          10522  2              ! and its value.  Note that we set the radix to decimal over this call
;10431          10523  2              ! and then restore it.  Also note that the Expression Parser sets
;10432          10524  2              ! TERMINATOR_CODE and TERMINATOR_LENGTH as a side-effect.
;10433          10525  2              !
;10434          10526  2              SAVED_RADIX = .EXPRESSION_RADIX;
;10435          10527  2              EXPRESSION_RADIX = DBG$K_DECIMAL;
;10436          10528  2              VALPTR = DBG$EXPRESSION_PARSER (FALSE, SET_CONSTANT_TERM_TBL);
;10437          10529  2              EXPRESSION_RADIX = .SAVED_RADIX;
;10438          10530  2
;10439          10531  2
;10440          10532  2              ! Check the terminator code.  If there was no terminator (the input
;10441          10533  2              ! line just ended), signal an error.  Otherwise we got a comma or clos-
;10442          10534  2              ! ing subscript parenthesis and we increment CHARPTR to get past it.
;10443          10535  2              !
;10444          10536  2              IF .TERMINATOR_CODE EQL TOKEN$K_TERM_NONE THEN SIGNAL(DBG$_MISCLOSUB);
;10445          10537  2              CHARPTR = .CHARPTR + .TERMINATOR_LENGTH;
;10446          10538  2
;10447          10539  2
;10448          10540  2              ! Create a set constant value descriptor.  Its type is the type of
;10449          10541  2              ! the first set constant data type.
;10450          10542  2              !
;10451          10543  2              IF .CREATE
;10452          10544  2              THEN
;10453          10545  3                  BEGIN
;10454          10546  3                  SETVALPTR = DBG$MAKE_SKELETON_DESC(DBG$K_VALUE_DESC, 8*4);
;10455          10547  3                  SETVALPTR[DBG$B_DHDR_LANG] = .DBG$GB_LANGUAGE;
;10456          10548  3                  SETVALPTR[DBG$B_DHDR_KIND] = RST$K_DATA;
;10457          10549  3                  SETVALPTR[DBG$B_DHDR_FCODE] = RST$K_TYPE_SET;
;10458          10550  3                  SETVALPTR[DBG$W_VALUE_LENGTH] = 32;
;10459          10551  3                  SETVALPTR[DBG$L_VALUE_POINTER] = SETVALPTR[DBG$A_VALUE_ADDRESS];
;10460          10552  3                  SETVAL = .SETVALPTR[DBG$L_VALUE_POINTER];
;10461          10553  3                  SELECTONE .VALPTR[DBG$B_DHDR_FCODE] OF
;10462          10554  3                      SET
;10463          10555  3                      [RST$K_TYPE_ATOMIC, RST$K_TYPE_DESCR]:
;10464          10556  4                          BEGIN
;10465          10557  4                          SETVALPTR[DBG$L_DHDR_TYPEID] = DBG$TYPEID_FOR_SET(
;10466          10558  4                              .VALPTR[DBG$B_VALUE_DTYPE], RST$K_TYPE_SET, 256, TRUE);
;10467          10559  3                          END;
;10468          10560  3
;10469          10561  3                      [RST$K_TYPE_ENUM]:
;10470          10562  4                          BEGIN
```

```
:10471          10563  5                TYPEID = .VALPTR[DBG$L_DHDR_TYPEID];
:10472          10564  5                SETVALPTR[DBG$L_DHDR_TYPEID] = DBG$TYPEID_FOR_SET(
:10473          10565  5                    .TYPEID[RST$L_DSTPTR], RST$K_TYPE_SET, 256);
:10474          10566  4                END;
:10475          10567  4
:10476          10568  4            [OTHERWISE]:
:10477          10569  4                SIGNAL(DBG$_ILLTYPE);
:10478          10570  4
:10479          10571  4            TES;
:10480          10572  4
:10481          10573  4        CREATE = FALSE;
:10482          10574  3        END;
:10483          10575  3
:10484          10576  3
:10485          10577  3    ! Check that the type of the set item is consistent with the
:10486          10578  3    ! type of the set as a whole.
:10487          10579  3    !
:10488          10580  3    IF NOT DBG$PERFORM_TYPEID_CHECK (ORT$K_TYPEID_SET_SET,
:10489          10581  3            .VALPTR, .SETVALPTR)
:10490          10582  3    THEN
:10491          10583  3        SIGNAL (DBG$_ILLSETCON);
:10492          10584  3
:10493          10585  3    VALADDR = .VALPTR[DBG$L_VALUE_POINTER];
:10494          10586  3
:10495          10587  3
:10496          10588  3    ! If the terminator at the end of this set constant expression was a dot
:10497          10589  3    ! we have a subrange (for example, "(1..5, 8)").  We thus set
:10498          10590  3    ! the subrange-range flag and save the low value of the range, i.e.
:10499          10591  3    ! the value we just picked up.
:10500          10592  3    !
:10501          10593  3    IF .TERMINATOR_CODE EQL TOKEN$K_TERM_DOT
:10502          10594  3    THEN
:10503          10595  4        BEGIN
:10504          10596  4        IF .THIS_SUBSCR_IS_RANGE THEN SIGNAL(DBG$_INVRANSPEC);
:10505          10597  4        THIS_SUBSCR_IS_RANGE = TRUE;
:10506          10598  4        LOW_RANGE_VAL = .VALADDR[0];
:10507          10599  4        END
:10508          10600  4
:10509          10601  4
:10510          10602  4    ! The terminator was not a dot, so we now have the full set constant
:10511          10603  4    ! specification.
:10512          10604  4    !
:10513          10605  3    ELSE
:10514          10606  4        BEGIN
:10515          10607  4
:10516          10608  4
:10517          10609  4        ! If this set constant is specified as a subrange, check that
:10518          10610  4        ! the first value in the range is not greater than the second.
:10519          10611  4        ! Also clear the subscript-is-range flag for the next subscript.
:10520          10612  4        !
:10521          10613  4        IF .THIS_SUBSCR_IS_RANGE
:10522          10614  4        THEN
:10523          10615  5            BEGIN
:10524          10616  5            IF .LOW_RANGE_VAL GTR .VALADDR[0] THEN SIGNAL(DBG$_INVRANSPEC);
:10525          10617  5            THIS_SUBSCR_IS_RANGE = FALSE;
:10526          10618  5            END
:10527          10619  5
```

```
;10528          10620  5
;10529          10621  5                    ! Otherwise, set the low range value to be the set constant value.
;10530          10622  5                    !
;10531          10623  4                    ELSE
;10532          10624  4                        LOW_RANGE_VAL = .VALADDR[0];
;10533          10625  4
;10534          10626  3                    END;
;10535          10627  3
;10536          10628  3
;10537          10629  3                ! Set the set value.
;10538          10630  3                !
;10539          10631  3                IF .LOW_RANGE_VAL LSS 0 OR .LOW_RANGE_VAL GTR 256                    5
;10540          10632  3                THEN
;10541          10633  3                    SIGNAL(DBG$_BITRANGE);                                           5
;10542          10634  3
;10543          10635  3                IF .VALADDR[0] LSS 0 OR .VALADDR[0] GTR 256
;10544          10636  3                THEN
;10545          10637  3                    SIGNAL(DBG$_BITRANGE);
;10546          10638  3
;10547          10639  3                INCR I FROM .LOW_RANGE_VAL TO .VALADDR[0] DO
;10548          10640  3                    SETVAL[.I] = TRUE;
;10549          10641  2                END;                            ! End of WHILE loop over set constants
;10550          10642  2
;10551          10643  2
;10552          10644  2            ! We have picked up all the set constants within this set parentheses.
;10553          10645  2            ! Set Constant Value Descriptor is created.
;10554          10646  2            !
;10555          10647  2            RETURN .SETVALPTR;
;10556          10648  2
;10557          10649  1            END;
; INFO#250                  L1:10616
; Referenced LOCAL symbol LOW_RANGE_VAL is probably not initialized


                                        0FFC 00000 GET_SET_CONSTANT:
                                                   .WORD   Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11          ; 1044
                5D    8F 00000000'   FF  91 00002         CMPB    @CHARPTR, #93                        ; 1049
                                     30  12 0000A         BNEQ    2$                                   ;
                                     20  DD 0000C         PUSHL   #32                                  ; 1049
                      7E       7A    8F  9A 0000E         MOVZBL  #122, -(SP)                          ;
00000000G             00             02  FB 00012         CALLS   #2, DBG$MAKE_SKELETON_DESC           ;
                      52             50  D0 00019         MOVL    R0, SETVALPTR                        ;
                03    A2             01  8E 0001C         MNEGB   #1, 3(SETVALPTR)                     ; 1049
                06    A2       0608  8F  B0 00020         MOVW    #1544, 6(SETVALPTR)                  ; 1049
                18    A2         20  A2  9E 00026         MOVAB   32(R2), 24(SETVALPTR)                ; 1049
                14    A2 FFFF0020  8F  D0 0002B         MOVL    #-65504, 20(SETVALPTR)               ; 1050
                      00000000'    EF  D6 00033         INCL    CHARPTR                              ; 1050
                               0181  31  00039 1$:       BRW     22$                                  ; 1050
                      59           01  D0 0003C 2$:       MOVL    #1, CREATE                           ; 1051
                                   58  D4 0003F         CLRL    THIS_SUBSCR_IS_RANGE                 ; 1051
00000000'             EF           01  D0 00041         MOVL    #1, TERMINATOR_CODE                  ; 1051
                      02 00000000' EF  D1 00048 3$:       CMPL    TERMINATOR_CODE, #2                  ; 1051
                                   E8  13 0004F         BEQL    1$                                   ;
                      5B 00000000' EF  D0 00051         MOVL    EXPRESSION_RADIX, SAVED_RADIX        ; 1052
```

```
                00000000'  EF                  0A  D0 00058          MOVL    #10, EXPRESSION_RADIX                   ; 1052
                    00000000'  EF               9F  9F 0005F          PUSHAB  SET_CONSTANT_TERM_TBL                   ; 1052
                               7E               D4  D4 00065          CLRL    -(SP)                                   ;
                    D42A       CF               02  FB 00067          CALLS   #2, DBG$EXPRESSION_PARSER               ;
                               53               50  D0 0006C          MOVL    R0, VALPTR                              ;
                00000000'  EF                   5B  D0 0006F          MOVL    SAVED_RADIX, EXPRESSION_RADIX           ; 1052
                    00000000'  EF               D5  D5 00076          TSTL    TERMINATOR_CODE                         ; 1053
                               0D               12  12 0007C          BNEQ    4$                                      ;
                    00028E90   8F               DD  DD 0007E          PUSHL   #167568                                 ;
00000000G       00             01               FB  FB 00084          CALLS   #1, LIB$SIGNAL                          ;
00000000'       EF  00000000'  EF               C0  C0 0008B  4$:       ADDL2   TERMINATOR_LENGTH, CHARPTR              ; 1053
                    7E         59               E9  E9 00096          BLBC    CREATE, 9$                              ; 1054
                               20               DD  DD 00099          PUSHL   #32                                     ; 1054
                    7E         7A               8F  9A 0009B          MOVZBL  #122, -(SP)                             ;
00000000G       00             02               FB  FB 0009F          CALLS   #2, DBG$MAKE_SKELETON_DESC              ;
                               52               50  D0 000A6          MOVL    R0, SETVALPTR                           ;
                03         A2  00000000G        00  90 000A9          MOVB    DBG$GB_LANGUAGE, 3(SETVALPTR)           ; 1054
                06         A2      0608         8F  B0 000B1          MOVW    #1544, 6(SETVALPTR)                     ; 1054
                14         A2                   20  B0 000B7          MOVW    #32, 20(SETVALPTR)                      ; 1055
                18         A2          20       A2  9E 000BB          MOVAB   32(R2), 24(SETVALPTR)                   ; 1055
                           5A          18       A2  D0 000C0          MOVL    24(SETVALPTR), SETVAL                   ; 1055
                           50          06       A3  9A 000C4          MOVZBL  6(VALPTR), R0                           ; 1055
                           02                   50  91 000C8          CMPB    R0, #2                                  ; 1055
                                                1B  1F 000CB          BLSSU   5$                                      ;
                           03                   50  91 000CD          CMPB    R0, #3                                  ;
                                                16  1A 000D0          BGTRU   5$                                      ;
                                                01  DD 000D2          PUSHL   #1                                      ; 1055
                           7E        0100       8F  3C 000D4          MOVZWL  #256, -(SP)                             ;
                                                08  DD 000D9          PUSHL   #8                                      ;
                           7E          16       A3  9A 000DB          MOVZBL  22(VALPTR), -(SP)                       ; 1055
00000000G       00                   04  FB 000DF          CALLS   #4, DBG$TYPEID_FOR_SET                  ;
                                     1A  11 000E6          BRB     6$                                      ;
                           04                   50  91 000E8  5$:       CMPB    R0, #4                                  ; 1056
                                                1B  12 000EB          BNEQ    7$                                      ;
                           56          08       A3  D0 000ED          MOVL    8(VALPTR), TYPEID                       ; 1056
                           7E        0100       8F  3C 000F1          MOVZWL  #256, -(SP)                             ; 1056
                                                08  DD 000F6          PUSHL   #8                                      ;
                                       0C       A6  DD 000F8          PUSHL   12(TYPEID)                              ; 1056
00000000G       00                   03  FB 000FB          CALLS   #3, DBG$TYPEID_FOR_SET                  ;
                08         A2                   50  D0 00102  6$:       MOVL    R0, 8(SETVALPTR)                        ;
                                                0D  11 00106          BRB     8$                                      ; 1055
                    000287D8   8F               DD  DD 00108  7$:       PUSHL   #165848                                 ; 1056
00000000G       00                   01  FB 0010E          CALLS   #1, LIB$SIGNAL                          ;
                                     59  D4 00115  8$:       CLRL    CREATE                                  ; 1057
                                     52  DD 00117  9$:       PUSHL   SETVALPTR                               ; 1058
                                     53  DD 00119          PUSHL   VALPTR                                  ;
                                     02  DD 0011B          PUSHL   #2                                      ; 1058
00000000G       00                   03  FB 0011D          CALLS   #3, DBG$PERFORM_TYPEID_CHECK            ;
                                     50  E8 00124          BLBS    R0, 10$                                 ;
                    00028F20   8F               DD  DD 00127          PUSHL   #167712                                 ; 1058
00000000G       00                   01  FB 0012D          CALLS   #1, LIB$SIGNAL                          ;
                           57          18       A3  D0 00134  10$:       MOVL    24(VALPTR), VALADDR                     ; 1058
                           0F  00000000'        EF  D1 00138          CMPL    TERMINATOR_CODE, #15                    ; 1059
                                                15  12 0013F          BNEQ    12$                                     ;
                           0D                   58  E9 00141          BLBC    THIS_SUBSCR_IS_RANGE, 11$               ; 1059
                    00028F08   8F               DD  DD 00144          PUSHL   #167688                                 ;
00000000G       00                   01  FB 0014A          CALLS   #1, LIB$SIGNAL                          ;
```

```
                        58              01  D0 00151 11$:   MOVL    #1, THIS_SUBSCR_IS_RANGE                ; 1059
                                        1C  11 00154         BRB     1$                                      ; 1059
                        19              58  E9 00156 12$:   BLBC        SUBSCR_IS_RANGE, 14$                ; 1061
                        54              67  D0 00159         MOVL      ALADDR), R4                           ; 1061
                        54              55  D1 0015C         CMPL    LOW_RANGE_VAL, R4
                                        0D  15 0015F         B       13$
                    00028F08            8F  DD 00161         P       #167688
                00000000G   00              01  FB 00167         CALLS   #1, LIB$SIGNAL
                                        58  D4 0016E 13$:    CLRL    THIS_SUBSCR_IS_RANGE                    ; 1061
                                        06  11 00170         BRB     15$                                     ; 1061
                        54              67  D0 00172 14$:    MOVL    (VALADDR), R4                           ; 1062
                        55              54  D0 00175         MOVL    R4, LOW_RANGE_VAL
                                        55  D5 00178 15$:    TSTL    LOW_RANGE_VAL                           ; 1063
                                        09  19 0017A         BLSS    16$
                00000100    8F              55  D1 0017C         CMPL    LOW_RANGE_VAL, #256
                                        0D  15 00183         BLEQ    17$
                    00028248            8F  DD 00185 16$:    PUSHL   #164424                                 ; 1063
                00000000G   00              01  FB 0018B         CALLS   #1, LIB$SIGNAL
                                        54  D5 00192 17$:    TSTL    R4                                      ; 1063
                                        09  19 00194         BLSS    18$
                00000100    8F              54  D1 00196         CMPL    R4, #256
                                        0D  15 0019D         BLEQ    19$
                    00028248            8F  DD 0019F 18$:    PUSHL   #164424                                 ; 1063
                00000000G   00              01  FB 001A5         CALLS   #1, LIB$SIGNAL
                        50      FF      A5  9E 001AC 19$:    MOVAB   -1(R5), I                               ; 1064
                                        04  11 001B0         BRB     21$
                    00              6A      50  E2 001B2 20$:    BBSS    I, (SETVAL), 21$
                    F8              50      54  F3 001B6 21$:    AOBLEQ  R4, I, 20$
                                        FE8B    31 001BA         BRW     3$                                      ; 1051
                        50              52  D0 001BD 22$:    MOVL    SETVALPTR, R0                           ; 1064
                                            04 001C0         RET                                             ; 1064

; Routine Size:  449 bytes,    Routine Base:  DBG$CODE + 3162
```

```
;10559      10650  1     ROUTINE GET_SUBSCRIPTS(PRIMPTR): NOVALUE =
;10560      10651  1     !
;10561      10652  1     ! FUNCTION
;10562      10653  1     !       This routine picks up subscript values in an array reference.  It
;10563      10654  1     !       calls DBG$EXPRESSION_PARSER to parse and evaluate each subscript
;10564      10655  1     !       expression.  It also checks the data type of each subscript value
;10565      10656  1     !       and converts it to the appropriate data type as necessary.  The
;10566      10657  1     !       ultimate subscript values are stored as integers in the Primary
;10567      10658  1     !       Descriptor Array Sub-Node for the array being subscripted.  A new
;10568      10659  1     !       Sub-Node for the array element type is then appended so that the
;10569      10660  1     !       Primary Descriptor becomes a descriptor for the array element
;10570      10661  1     !       selected by the subscripting.
;10571      10662  1     !
;10572      10663  1     !       This routine also handles subscript ranges, such as ARR(1:5,3:10).
;10573      10664  1     !       It does so by modifying the subscript lower and upper bounds in
;10574      10665  1     !       the Array Sub-Node subscript vector to in effect define a new
;10575      10666  1     !       array, namely the array "slice" defined by the subscript ranges.
;10576      10667  1     !       In this case no Sub-Node for the array element is appended since
;10577      10668  1     !       the Primary Descriptor still defines an array.
;10578      10669  1     !
;10579      10670  1     !       This routine assumes that the opening subscript parenthesis has
;10580      10671  1     !       already been found and that the parse pointer points to the start
;10581      10672  1     !       of the first subscript expression.  When this routine returns,
;10582      10673  1     !       the parse pointer is left pointing at the first character after
;10583      10674  1     !       the closing subscript parenthesis.
;10584      10675  1     !
;10585      10676  1     ! INPUTS
;10586      10677  1     !       PRIMPTR - A pointer to the Primary Descriptor for an array about
;10587      10678  1     !                 to be subscripted.
;10588      10679  1     !
;10589      10680  1     ! OUTPUTS
;10590      10681  1     !       The PRIMPTR Primary Descriptor is changed to include the subscript
;10591      10682  1     !                 information (the actual subscript values) and a new Sub-
;10592      10683  1     !                 Node for the selected array element.  PRIMPTR itself is
;10593      10684  1     !                 not changed, however.
;10594      10685  1     !
;10595      10686  1
;10596      10687  2         BEGIN
;10597      10688  2
;10598      10689  2         MAP
;10599      10690  2             PRIMPTR: REF DBG$PRIMARY;          ! Pointer to array Primary Descriptor
;10600      10691  2
;10601      10692  2         LOCAL
;10602      10693  2             BITSIZE,                           ! Bit size of subscript value data type
;10603      10694  2             CHECK_VAL,
;10604      10695  2             DECLTYPE: REF DBG$VALDESC,         ! Pointer to Value Descriptor for
;10605      10696  2                                                !      declared subscript data type
;10606      10697  2             DESCR: DBG$STG_DESC,               ! String descriptor
;10607      10698  2             DSCADDR: REF DBG$STG_DESC,         ! Pointer to a string descriptor
;10608      10699  2             FCODE,                             ! Data type FCODE for array element type
;10609      10700  2             LA_PTR: REF VECTOR[,BYTE],         ! Lookahead pointer into input
;10610      10701  2             LOW_RANGE_VAL,                     ! Low value of a subscript range
;10611      10702  2             NODEPTR: REF DBG$PRIM_NODE,        ! Pointer to Prim Descr Array Sub-Node
;10612      10703  2             SAVED_RADIX,                       ! Temporarily saved expression radix
;10613      10704  2             SUBSCR_COUNT,                      ! Actual subscript count in input line
;10614      10705  2             SUBVECTOR:                         ! Pointer to subscript block-vector
;10615      10706  2                     REF DBG$PRIM_NODE_SUBS,    !      in Primary Descr Array Sub-Node
```

```
;10616      10707  2                THIS_SUBSCR_IS_RANGE,          ! Flag set if the current subscript is
;10617      10708  2                                               !      given as a subscript range
;10618      10709  2                TOKEN,                         ! Lexical Token
;10619      10710  2                TYPECODE,                      ! VAX standard type code for atomic type
;10620      10711  2                TYPEID: REF RST$ENTRY,         ! Holds a typeid
;10621      10712  2                VALADDR: REF VECTOR[,LONG],    ! Pointer to integer subscript value
;10622      10713  2                VALPTR: REF DBG$VALDESC;       ! Pointer to subscript Value Descriptor
;10623      10714  2
;10624      10715  2            DBG$GL_CURRENT_PRIMARY = .PRIMPTR;
;10625      10716  2
;10626      10717  2
;10627      10718  2            ! Check that the Primary Descriptor is really a Primary Descriptor!
;10628      10719  2            !
;10629      10720  2            IF .PRIMPTR[DBG$B_DHDR_TYPE] NEQ DBG$K_PRIMARY_DESC
;10630      10721  2            THEN
;10631      10722  2                SIGNAL(DBG$_NOTARRAY);
;10632      10723  2
;10633      10724  2
;10634      10725  2            ! Check that the Primary Descriptor is for an array--otherwise subscripting
;10635      10726  2            ! is not allowed.
;10636      10727  2            !
;10637      10728  2            IF .PRIMPTR[DBG$B_DHDR_FCODE] NEQ RST$K_TYPE_ARRAY
;10638      10729  2            THEN
;10639      10730  3                BEGIN
;10640      10731  3
;10641      10732  3                ! Check for the possibility of a substring reference.
;10642      10733  3                ! E.g., in FORTRAN, if a variable is declared CHARACTER*n
;10643      10734  3                ! we see it in the DST as an atomic item of type T.
;10644      10735  3                ! We want to allow X(i:j) in this case.
;10645      10736  3                !
;10646      10737  3                IF .PRIMPTR[DBG$B_DHDR_FCODE] EQL RST$K_TYPE_ATOMIC
;10647      10738  3                THEN
;10648      10739  4                    BEGIN
;10649      10740  4                    DBG$STA_TYP_ATOMIC (.PRIMPTR[DBG$L_DHDR_TYPEID], TYPECODE, BITSIZE);
;10650      10741  4
;10651      10742  4
;10652      10743  4                    ! Special case for BASIC.  In this case, each element of an
;10653      10744  4                    ! array is a VMS descriptor.
;10654      10745  4                    !
;10655      10746  4                    IF .TYPECODE EQL DSC$K_DTYPE_DSC
;10656      10747  4                    THEN
;10657      10748  5                        BEGIN
;10658      10749  5                        DBG$PRIM_TO_VAL (.PRIMPTR, DBG$K_VALUE_DESC, VALPTR);
;10659      10750  5                        DSCADDR = VALPTR[DBG$A_VALUE_VMSDESC];
;10660      10751  5                        IF .DSCADDR[DSC$B_DTYPE] EQL DSC$K_DTYPE_T
;10661      10752  5                        THEN
;10662      10753  6                            BEGIN
;10663      10754  6                            GET_SUBSTRING (.PRIMPTR, .DSCADDR);
;10664      10755  6                            RETURN;
;10665      10756  5                            END;
;10666      10757  4                        END;
;10667      10758  4
;10668      10759  4                    IF .TYPECODE EQL DSC$K_DTYPE_T
;10669      10760  4                    THEN
;10670      10761  5                        BEGIN
;10671      10762  5                        DESCR[DSC$B_CLASS] = DSC$K_CLASS_S;
;10672      10763  5                        DESCR[DSC$B_DTYPE] = DSC$K_DTYPE_T;
```

```
;10673          10764  5                DESCR[DSC$W_LENGTH] = .BITSIZE/8;
;10674          10765  5                GET_SUBSTRING (.PRIMPTR, DESCR);
;10675          10766  5                RETURN;
;10676          10767  4                END;
;10677          10768  3            END;
;10678          10769  3
;10679          10770  3        IF .PRIMPTR[DBG$B_DHDR_FCODE] EQL RST$K_TYPE_DESCR
;10680          10771  3        THEN
;10681          10772  4            BEGIN
;10682          10773  4            DBG$STA_TYP_DESCR (.PRIMPTR [DBG$L_DHDR_TYPEID], DSCADDR);
;10683          10774  4            IF .DSCADDR[DSC$B_DTYPE] EQL DSC$K_DTYPE_T
;10684          10775  4            THEN
;10685          10776  5                BEGIN
;10686          10777  5                GET_SUBSTRING (.PRIMPTR, .DSCADDR);
;10687          10778  5                RETURN;
;10688          10779  4                END;
;10689          10780  3            END;
;10690          10781  3
;10691          10782  3
;10692          10783  3        ! For typed pointers there are two special cases:
;10693          10784  3        ! For language C, subscripting of pointers is allowed; e.g.,
;10694          10785  3        ! PTR[n]  is equivalent to  *(PTR+n)
;10695          10786  3        ! For ADA, the pointer dereference is implicit; thus if PTR
;10696          10787  3        ! is a pointer then PTR(I) in ADA is equivalent to PTR^[I]
;10697          10788  3        ! in PASCAL.
;10698          10789  3        !
;10699          10790  3        IF .PRIMPTR[DBG$B_DHDR_FCODE] EQL RST$K_TYPE_TPTR
;10700          10791  3        THEN
;10701          10792  4            BEGIN
;10702          10793  4            CASE .DBG$GB_LANGUAGE FROM DBG$K_MIN_LANGUAGE TO DBG$K_MAX_LANGUAGE OF
;10703          10794  4                SET
;10704          10795  4
;10705          10796  4                ! For language C, subscripting of pointers is allowed; e.g.,
;10706          10797  4                ! PTR[n]  is equivalent to  *(PTR+n)
;10707          10798  4                !
;10708          10799  4                [DBG$K_C]:
;10709          10800  5                    BEGIN
;10710          10801  5                    LOCAL
;10711          10802  5                        ADDRESS,                ! Address of array (value of pointer)
;10712          10803                          DUMMY,                  ! Dummy output parameter
;10713          10804  5                        VALPTR: REF DBG$VALDESC;! Pointer to a Value Descriptor
;10714          10805  5
;10715          10806  5
;10716          10807  5                    BUILTIN
;10717          10808  5                        REMQUE;
;10718          10809  5
;10719          10810  5
;10720          10811  5                    ! Compute the value of the pointer that is represented
;10721          10812  5                    ! by the current primary.
;10722          10813  5                    !
;10723          10814  5                    DBG$PRIM_TO_VAL (.PRIMPTR, DBG$K_VALUE_DESC, VALPTR);
;10724          10815  5                    ADDRESS = .VALPTR[DBG$L_VALUE_VALUE0];
;10725          10816  5
;10726          10817  5
;10727          10818  5                    ! Unlink the existing subnode and build a new one to
;10728          10819  5                    ! describe an array.
;10729          10820  5                    !
```

```
;10730      10821  5          NODEPTR = .PRIMPTR[DBG$L_PRIM_BLINK];
;10731      10822  5          REMQUE(.NODEPTR, DUMMY);
;10732      10823  5          TYPEID = DBG$TYPEID_FOR_ARRAY(.PRIMPTR[DBG$L_DHDR_TYPEID],
;10733      10824  5                                        .ADDRESS);
;10734      10825  5          DBG$BUILD_PRIMARY_SUBNODE (.PRIMPTR, RST$K_DATA, 0,
;10735      10826  5                                     RST$K_TYPE_ARRAY, .TYPEID, 0);
;10736      10827  5          NODEPTR = .PRIMPTR[DBG$L_PRIM_BLINK];
;10737      10828  5          NODEPTR[DBG$L_PNODE_RELOC] = .ADDRESS;
;10738      10829  4          END;
;10739      10830  4
;10740      10831  4
;10741      10832  4      ! For ADA, the pointer dereference is implicit; thus if PTR
;10742      10833  4      ! is a pointer then PTR(I) in ADA is equivalent to PTR^[I]
;10743      10834  4      ! in PASCAL.
;10744      10835  4      !
;10745      10836  4      [DBG$K_ADA]:
;10746      10837  5          BEGIN
;10747      10838  5
;10748      10839  5
;10749      10840  5          ! Dereference the pointer.
;10750      10841  5          !
;10751      10842  5          WHILE .PRIMPTR[DBG$B_DHDR_FCODE] EQL RST$K_TYPE_TPTR DO
;10752      10843  5              GET_DEREFERENCE(.PRIMPTR);
;10753      10844  5
;10754      10845  5
;10755      10846  5          ! If the Primary is still not an array then
;10756      10847  5          ! signal an error.
;10757      10848  5          !
;10758      10849  5          IF .PRIMPTR[DBG$B_DHDR_FCODE] NEQ RST$K_TYPE_ARRAY
;10759      10850  5          THEN
;10760      10851  5              SIGNAL(DBG$_NOTARRAY);
;10761      10852  4          END;
;10762      10853  4
;10763      10854  4
;10764      10855  4              ! For other languages, we cannot subscript typed pointers.
;10765      10856  4              !
;10766      10857  4              [INRANGE]:
;10767      10858  4                  SIGNAL(DBG$_NOTARRAY);
;10768      10859  4
;10769      10860  4
;10770      10861  4              ! We do not expect any other language codes.
;10771      10862  4              !
;10772      10863  4              [OUTRANGE]:
;10773      10864  4                  $DBG_ERRCR('DBGPARSER\GET_SUBSCRIPTS 5');
;10774      10865  4
;10775      10866  4              TES;
;10776      10867  4          END
;10777      10868  4
;10778      10869  4
;10779      10870  4      ! Else the variable is neither a string nor a typed pointer
;10780      10871  4      ! nor an array so subscripting is not allowed.
;10781      10872  4      !
;10782      10873  3      ELSE
;10783      10874  3          SIGNAL(DBG$_NOTARRAY);
;10784      10875  2      END;
;10785      10876  2
;10786      10877  2
```

```
;10787      10878  2          ! Set up pointers to the Primary Descriptor Array Sub-Node and to the
;10788      10879  2          ! subscript vector within that node.
;10789      10880  2          !
;10790      10881  2          NODEPTR = .PRIMPTR[DBG$L_PRIM_BLINK];
;10791      10882  2          SUBVECTOR = NODEPTR[DBG$A_PNARR_SVECTOR];
;10792      10883  2          IF .NODEPTR[DBG$B_PNODE_FCODE] NEQ RST$K_TYPE_ARRAY
;10793      10884  2          THEN
;10794      10885  2              $DBG_ERROR('DBGPARSER\GET_SUBSCRIPTS 10');
;10795      10886  2
;10796      10887  2
;10797      10888  2          ! Loop through the subscript expressions for this array reference.  Each
;10798      10889  2          ! subscript is parsed, evaluated, and converted to the appropriate type
;10799      10890  2          ! (with the type being checked in the process).  It is then checked for
;10800      10891  2          ! being in range and its integer value is stored in the Array Sub-Node's
;10801      10892  2          ! subscript block-vector.  Note that TERMINATOR_CODE is set within the
;10802      10893  2          ! loop as a side-effect of the call on DBG$EXPRESSION_PARSER.
;10803      10894  2          !
;10804      10895  2          THIS_SUBSCR_IS_RANGE = FALSE;
;10805      10896  2          SUBSCR_COUNT = .NODEPTR[DBG$B_PNARR_SUBCNT];
;10806      10897  2          TERMINATOR_CODE = TOKEN$K_TERM_COMMA;
;10807      10898  2          WHILE .TERMINATOR_CODE NEQ TOKEN$K_TERM_CLOSE DO
;10808      10899  3              BEGIN
;10809      10900  3
;10810      10901  3
;10811      10902  3              ! Check that the actual subscript count does not exceed the dimension
;10812      10903  3              ! count for the array.
;10813      10904  3              !
;10814      10905  3              IF .SUBSCR_COUNT GEQ .NODEPTR[DBG$B_PNARR_DIMCNT]
;10815      10906  3              THEN
;10816      10907  3                  SIGNAL(DBG$_TOOMANSUB, 1, .NODEPTR[DBG$B_PNARR_DIMCNT]);
;10817      10908  3
;10818      10909  3
;10819      10910  3              ! Look for the asterisk. X(*) is the same as X(lower:upper).
;10820      10911  3              ! If we find the asterisk then advance the character pointer beyond
;10821      10912  3              ! the asterisk and also increment the subscript count.
;10822      10913  3              !
;10823      10914  3              LA_PTR = .CHARPTR;
;10824      10915  3              WHILE .LA_PTR[0] EQL ' ' DO LA_PTR = .LA_PTR + 1;
;10825      10916  3              IF .LA_PTR[0] EQL '*'
;10826      10917  3              THEN
;10827      10918  4                  BEGIN
;10828      10919  4                  CHARPTR = .LA_PTR + 1;
;10829      10920  4
;10830      10921  4
;10831      10922  4                  ! Call the Lexical Scanner to take us past the ',' or
;10832      10923  4                  ! or ']' or ')'. This will set TERMINATOR_CODE to the
;10833      10924  4                  ! terminator that is seen. If we do not see a terminator
;10834      10925  4                  ! then signal a syntax error. Also signal an error if
;10835      10926  4                  ! ':' was the terminator.
;10836      10927  4                  !
;10837      10928  4                  TOKEN = DBG$LEXICAL_SCANNER (FALSE, FALSE,
;10838      10929  4                                               .SUBSCRIPT_TERM_TBL, 0);
;10839      10930  4                  IF .TOKEN NEQ TERMINATOR_TOKEN
;10840      10931  4                  THEN
;10841      10932  5                      BEGIN
;10842      10933  5                      LOCAL
;10843      10934  5                          ASCIC_STRING: VECTOR[2,BYTE];
```

```
;10844      10935  5                ASCIC_STRING[0] = 1;
;10845      10936  5                ASCIC_STRING[1] = .CHARPTR[0];
;10846      10937  5                SIGNAL(DBG$_SYNERREXPR, 1, ASCIC_STRING);
;10847      10938  4                END;
;10848      10939  4            IF .TERMINATOR_CODE EQL TOKEN$K_TERM_COLON
;10849      10940  4            THEN
;10850      10941  4                SIGNAL (DBG$_INVRANSPEC);
;10851      10942  4            IF .TERMINATOR_CODE EQL TOKEN$K_TERM_NONE
;10852      10943  4            THEN
;10853      10944  4                SIGNAL(DBG$_MISCLOSUB);
;10854      10945  4            CHARPTR = .CHARPTR + .TERMINATOR_LENGTH;
;10855      10946  4
;10856      10947  4
;10857      10948  4            ! Turn this reference into a range.
;10858      10949  4            ! If it was not already a range, then turn all previous
;10859      10950  4            ! subscripts into ranges.
;10860      10951  4            !
;10861      10952  4            IF NOT .NODEPTR[DBG$V_PNARR_RANGE]
;10862      10953  4            THEN
;10863      10954  5                BEGIN
;10864      10955  5                NODEPTR[DBG$V_PNARR_RANGE] = TRUE;
;10865      10956  5                INCR I FROM 0 TO .SUBSCR_COUNT - 1 DO
;10866      10957  6                    BEGIN
;10867      10958  6                    SUBVECTOR[.I, DBG$L_PNSUB_LBOUND] =
;10868      10959  6                                        .SUBVECTOR[.I, DBG$L_PNSUB_SVALUE];
;10869      10960  6                    SUBVECTOR[.I, DBG$L_PNSUB_UBOUND] =
;10870      10961  6                                        .SUBVECTOR[.I, DBG$L_PNSUB_SVALUE];
;10871      10962  5                    END;
;10872      10963  5
;10873      10964  4                END;
;10874      10965  4                SUBSCR_COUNT = .SUBSCR_COUNT + 1;
;10875      10966  4                NODEPTR[DBG$V_PNARR_RANGE] = TRUE;
;10876      10967  4                END
;10877      10968  4
;10878      10969  3            ELSE
;10879      10970  4                BEGIN
;10880      10971  4
;10881      10972  4
;10882      10973  4                ! Call the expression parser to pick up the next subscript expression
;10883      10974  4                ! and its value.  Note that we set the radix to decimal over this call
;10884      10975  4                ! and then restore it.  Also note that the Expression Parser sets
;10885      10976  4                ! TERMINATOR_CODE and TERMINATOR_LENGTH as a side-effect.
;10886      10977  4                !
;10887      10978  4                SAVED_RADIX = .EXPRESSION_RADIX;
;10888      10979  4                EXPRESSION_RADIX = DBG$K_DECIMAL;
;10889      10980  4                VALPTR = DBG$EXPRESSION_PARSER (FALSE, .SUBSCRIPT_TERM_TBL);
;10890      10981  4                EXPRESSION_RADIX = .SAVED_RADIX;
;10891      10982  4
;10892      10983  4
;10893      10984  4                ! Check the terminator code.  If there was no terminator (the input
;10894      10985  4                ! line just ended), signal an error.  Otherwise we got a comma or clos-
;10895      10986  4                ! ing subscript parenthesis and we increment CHARPTR to get past it.
;10896      10987  4                !
;10897      10988  4                IF .TERMINATOR_CODE EQL TOKEN$K_TERM_NONE THEN SIGNAL(DBG$_MISCLOSUB);
;10898      10989  4                CHARPTR = .CHARPTR + .TERMINATOR_LENGTH;
;10899      10990  4
;10900      10991  4
```

```
;10901      10992  4          ! We now need to convert the subscript to one of the appropriate
;10902      10993  4          ! dtype. We need to set up a target descriptor for the conversion
;10903      10994  4          ! routine. We allocate a skeleton descriptor and fill in some of
;10904      10995  4          ! the fields.
;10905      10996  4          !
;10906      10997  4          DECLTYPE = DBG$MAKE_SKELETON_DESC(DBG$K_VALUE_DESC, 4);
;10907      10998  4          DECLTYPE[DBG$B_DHDR_KIND] = RST$K_DATA;
;10908      10999  4          TYPEID = .SUBVECTOR[.SUBSCR_COUNT, DBG$L_PNSUB_TYPEID];
;10909      11000  4          DECLTYPE[DBG$L_DHDR_TYPEID] = .TYPEID;
;10910      11001  4          DECLTYPE[DBG$L_VALUE_POINTER] = DECLTYPE[DBG$A_VALUE_ADDRESS];
;10911      11002  4
;10912      11003  4
;10913      11004  4          ! We now must fill in the FCODE, CLASS, DTYPE, and LENGTH fields
;10914      11005  4          ! of the target descriptor. To do this, we need to look at the
;10915      11006  4          ! typeid.
;10916      11007  4          !
;10917      11008  4          IF .TYPEID EQL 0
;10918      11009  4          THEN
;10919      11010  5              BEGIN
;10920      11011  5
;10921      11012  5
;10922      11013  5              ! No typeid available - assume longword integer.
;10923      11014  5              !
;10924      11015  5              DECLTYPE[DBG$B_DHDR_FCODE] = RST$K_TYPE_ATOMIC;
;10925      11016  5              DECLTYPE[DBG$B_VALUE_CLASS] = DSC$K_CLASS_S;
;10926      11017  5              DECLTYPE[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_L;
;10927      11018  5              DECLTYPE[DBG$W_VALUE_LENGTH] = 4;
;10928      11019  5              END
;10929      11020  5
;10930      11021  4          ELSE
;10931      11022  5              BEGIN
;10932      11023  5
;10933      11024  5
;10934      11025  5              ! Typeid is available - determine FCODE from it.
;10935      11026  5              !
;10936      11027  5              DBG$STA_SYMSIZE(.TYPEID, BITSIZE);
;10937      11028  5              FCODE = .TYPEID[RST$B_FCODE];
;10938      11029  5              DECLTYPE[DBG$B_DHDR_FCODE] = .FCODE;
;10939      11030  5              IF .FCODE EQL RST$K_TYPE_ATOMIC
;10940      11031  5              THEN
;10941      11032  6                  BEGIN
;10942      11033  6
;10943      11034  6
;10944      11035  6                  ! Atomic data - determine class, dtype, and length from DST.
;10945      11036  6                  !
;10946      11037  6                  DBG$STA_TYP_ATOMIC(.TYPEID, TYPECODE, BITSIZE);
;10947      11038  6                  IF .TYPECODE EQL DST$K_BOOL
;10948      11039  6                  THEN
;10949      11040  6                      TYPECODE = DSC$K_DTYPE_TF;
;10950      11041  6                  DECLTYPE[DBG$B_VALUE_CLASS] = DBG$MAP_DTYPE_CLASS (
;10951      11042  6                      .TYPECODE, FALSE);
;10952      11043  6                  DECLTYPE[DBG$B_VALUE_DTYPE] = .TYPECODE;
;10953      11044  6                  IF  .TYPECODE NEQ DSC$K_DTYPE_V
;10954      11045  6                  AND .TYPECODE NEQ DSC$K_DTYPE_SV
;10955      11046  6                  AND .TYPECODE NEQ DSC$K_DTYPE_VU
;10956      11047  6                  AND .TYPECODE NEQ DSC$K_DTYPE_SVU
;10957      11048  6                  AND .TYPECODE NEQ DSC$K_DTYPE_TF
```

```
;10958      11049  6                THEN
;10959      11050  6                    DECLTYPE[DBG$W_VALUE_LENGTH] = (.BITSIZE+7)/8
;10960      11051  6                ELSE
;10961      11052  6                    DECLTYPE[DBG$W_VALUE_LENGTH] = .BITSIZE;
;10962      11053  6                END
;10963      11054  6
;10964      11055  5            ELSE IF .FCODE EQL RST$K_TYPE_DESCR
;10965      11056  5            THEN
;10966      11057  6                BEGIN
;10967      11058  6
;10968      11059  6                ! Descriptor data - determine class, dtype, and length from DST.
;10969      11060  6                !
;10970      11061  6                DBG$STA_TYP_DESCR (.TYPEID, DECLTYPE[DBG$A_VALUE_VMSDESC]);
;10971      11062  6                IF .DECLTYPE[DBG$B_VALUE_DTYPE] EQL DST$K_BOOL
;10972      11063  6                THEN
;10973      11064  6                    DECLTYPE[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_TF;
;10974      11065  6                END
;10975      11066  6
;10976      11067  5            ELSE
;10977      11068  6                BEGIN
;10978      11069  6
;10979      11070  6                ! Language-specific fcodes. Here we dummy in the dtype field
;10980      11071  6                ! with a special code. Determine class and length information
;10981      11072  6                ! using routines in DBGEVALOP.
;10982      11073  6                !
;10983      11074  6                DECLTYPE[DBG$B_VALUE_CLASS] = 0;
;10984      11075  6                DECLTYPE[DBG$B_VALUE_DTYPE] = 0;
;10985      11076  6                DECLTYPE[DBG$W_VALUE_LENGTH] = (.BITSIZE+7)/8;
;10986      11077  5                END;
;10987      11078  4            END;
;10988      11079  4
;10989      11080  4
;10990      11081  4            ! Finally call the conversion routine. This routine checks that
;10991      11082  4            ! the conversion is legal before doing it.
;10992      11083  4            !
;10993      11084  4            VALPTR = DBG$EVAL_LANG_OPERATOR(DBG$GL_CONVERT_TOKEN, .VALPTR, .DECLTYPE);
;10994      11085  4
;10995      11086  4
;10996      11087  4            ! Check that the subscript value is within the array bounds and give an
;10997      11088  4            ! informational message if not (execution continues after the message).
;10998      11089  4            ! For ADA, we make sure to check enumeration subscripts by their
;10999      11090  4            ! position and not by their value.
;11000      11091  4            !
;11001      11092  4            VALADDR = .VALPTR[DBG$L_VALUE_POINTER];
;11002      11093  4            IF (.DBG$GB_LANGUAGE EQL DBG$K_ADA) AND
;11003      11094  4               (.TYPEID NEQ 0) AND
;11004      11095  5               (.FCODE EQL RST$K_TYPE_ENUM)
;11005      11096  4            THEN
;11006      11097  4                CHECK_VAL = DBG$ENUM_POS(.TYPEID,.VALADDR[0])
;11007      11098  4            ELSE
;11008      11099  4                CHECK_VAL = .VALADDR[0];
;11009      11100  4
;11010      11101  4            IF (.CHECK_VAL LSS .SUBVECTOR[.SUBSCR_COUNT, DBG$L_PNSUB_LBOUND]) OR
;11011      11102  5               (.CHECK_VAL GTR .SUBVECTOR[.SUBSCR_COUNT, DBG$L_PNSUB_UBOUND])
;11012      11103  4            THEN
;11013      11104  4                SIGNAL(DBG$_SUBOUTBND, 4, .SUBSCR_COUNT + 1, .CHECK_VAL,
;11014      11105  4                                .SUBVECTOR[.SUBSCR_COUNT, DBG$L_PNSUB_LBOUND],
```

```
;11015          11106  4                    .SUBVECTOR[.SUBSCR_COUNT, DBG$L_PNSUB_UBOUND]);
;11016          11107  4
;11017          11108  4
;11018          11109  4            ! If the terminator at the end of this subscript expression was a colon
;11019          11110  4            ! we have a subscript range (for example, "ARR(1:5,2)").  We thus set
;11020          11111  4            ! the subscript-range flag and save the low value of the range, i.e.
;11021          11112  4            ! the value we just picked up.  If this is the first range in the
;11022          11113  4            ! subscript list, we also turn all previous subscripts into ranges by
;11023          11114  4            ! setting the lower and upper bound for each such subscript to the
;11024          11115  4            ! corresponding subscript value.  This in effect defines a new array
;11025          11116  4            ! which constitutes a "slice" of the original array.
;11026          11117  4            !
;11027          11118  4            IF .TERMINATOR_CODE EQL TOKEN$K_TERM_COLON
;11028          11119  4            THEN
;11029          11120  5                BEGIN                                                          ;
;11030          11121  5                IF .THIS_SUBSCR_IS_RANGE THEN SIGNAL(DBG$_INVRANSPEC);
;11031          11122  5                THIS_SUBSCR_IS_RANGE = TRUE;
;11032          11123  5                LOW_RANGE_VAL = .VALADDR[0];
;11033          11124  5                IF NOT .NODEPTR[DBG$V_PNARR_RANGE]
;11034          11125  5                THEN
;11035          11126  6                    BEGIN
;11036          11127  6                    NODEPTR[DBG$V_PNARR_RANGE] = TRUE;
;11037          11128  6                    INCR I FROM 0 TO .SUBSCR_COUNT - 1 DO
;11038          11129  7                        BEGIN
;11039          11130  7                        SUBVECTOR[.I, DBG$L_PNSUB_LBOUND] =
;11040          11131  7                                    .SUBVECTOR[.I, DBG$L_PNSUB_SVALUE];
;11041          11132  7                        SUBVECTOR[.I, DBG$L_PNSUB_UBOUND] =
;11042          11133  7                                    .SUBVECTOR[.I, DBG$L_PNSUB_SVALUE];
;11043          11134  6                        END;
;11044          11135  6
;11045          11136  5                    END;
;11046          11137  5
;11047          11138  5                 END
;11048          11139  5
;11049          11140  5
;11050          11141  5            ! The terminator was not a colon, so we now have the full subscript
;11051          11142  5            ! specification.  Fill the subscript value into the Array Sub-Node's
;11052          11143  5            ! subscript vector.  Set up the bounds for an array "slice" if this
;11053          11144  5            ! or any previous subscript specification in this array reference
;11054          11145  5            ! consisted of a subscript range.  Also bump the subscript count.
;11055          11146  5            !
;11056          11147  4            ELSE
;11057          11148  5                BEGIN
;11058          11149  5
;11059          11150  5
;11060          11151  5                    ! If this subscript is specified as a subscript range, check that
;11061          11152  5                    ! the first value in the range is not greater than the second.
;11062          11153  5                    ! Also clear the subscript-is-range flag for the next subscript.
;11063          11154  5                    !
;11064          11155  5                    IF .THIS_SUBSCR_IS_RANGE
;11065          11156  5                    THEN
;11066          11157  6                        BEGIN
;11067          11158  6                        IF .LOW_RANGE_VAL GTR .VALADDR[0] THEN SIGNAL(DBG$_INVRANSPEC);
;11068          11159  6                        THIS_SUBSCR_IS_RANGE = FALSE;
;11069          11160  6                        END
;11070          11161  6
;11071          11162  6
```

```
;11072     11163  6                    ! Otherwise, set the low range value to be the subscript value.
;11073     11164  6                    !
;11074     11165  5                    ELSE
;11075     11166  5                        LOW_RANGE_VAL = .VALADDR[0];
;11076     11167  5
;11077     11168  5
;11078     11169  5                    ! If this or any previous subscript in this array reference con-
;11079     11170  5                    ! tained a range specification (as in ARR(5:10)), then we arrange
;11080     11171  5                    ! the array's lower and upper bounds to reflect the array "slice"
;11081     11172  5                    ! the user is requesting.
;11082     11173  5                    !
;11083     11174  5                    IF .NODEPTR[DBG$V_PNARR_RANGE]
;11084     11175  5                    THEN
;11085     11176  6                        BEGIN
;11086     11177  6                        SUBVECTOR[.SUBSCR_COUNT, DBG$L_PNSUB_LBOUND] = .LOW_RANGE_VAL;
;11087     11178  6                        SUBVECTOR[.SUBSCR_COUNT, DBG$L_PNSUB_UBOUND] = .VALADDR[0];
;11088     11179  5                        END;
;11089     11180  5
;11090     11181  5
;11091     11182  5                    ! Finally fill in the subscript value itself (the start of the
;11092     11183  5                    ! range), increment the subscript count, and loop.
;11093     11184  5                    !
;11094     11185  5                    SUBVECTOR[.SUBSCR_COUNT, DBG$L_PNSUB_SVALUE] = .LOW_RANGE_VAL;
;11095     11186  5                    SUBSCR_COUNT = .SUBSCR_COUNT + 1;
;11096     11187  4                    END;
;11097     11188  3                END;
;11098     11189  2            END;                        ! End of WHILE loop over subscripts
;11099     11190  2
;11100     11191  2
;11101     11192  2            ! We have picked up all the subscripts within this set of subscript paren-
;11102     11193  2            ! theses.  Now check that the subscript count is the same as the dimension
;11103     11194  2            ! count unless the language allows fewer subscripts (as in Pascal where
;11104     11195  2            ! the array reference X[2,3][4] is valid).  If fewer subscripts are allowed
;11105     11196  2            ! we return now, leaving the Array Sub-Node at the end of the Primary.
;11106     11197  2            !
;11107     11198  2            NODEPTR[DBG$B_PNARR_SUBCNT] = .SUBSCR_COUNT;
;11108     11199  2            IF .SUBSCR_COUNT NEQ .NODEPTR[DBG$B_PNARR_DIMCNT]
;11109     11200  2            THEN
;11110     11201  3                BEGIN
;11111     11202  3                IF .MULTIPLE_SUBSCR THEN RETURN;
;11112     11203  3                SIGNAL(DBG$_TOOFEWSUB, 1, .NODEPTR[DBG$B_PNARR_DIMCNT]);
;11113     11204  2                END;
;11114     11205  2
;11115     11206  2
;11116     11207  2            ! If any subscript range was specified, we leave the Array Sub-Node at the
;11117     11208  2            ! end of the Primary even if all subscripts were specified.  In other words,
;11118     11209  2            ! we still have an array, namely the specified "array slice".
;11119     11210  2            !
;11120     11211  2            IF .NODEPTR[DBG$V_PNARR_RANGE] THEN RETURN;
;11121     11212  2
;11122     11213  2
;11123     11214  2            ! All the subscripts are specified.  Now set the EVAL bit in the Array
;11124     11215  2            ! Sub-Node to indicate that subscripting actually is being done.  Also
;11125     11216  2            ! construct a new Sub-Node for the array element type and append it to
;11126     11217  2            ! the Primary Descriptor.  Then return.
;11127     11218  2            !
;11128     11219  2            NODEPTR[DBG$V_PNODE_EVAL] = TRUE;
```

```
;11129          11220  2            FCODE = DBG$STA_TYPEFCODE(.NODEPTR[DBG$L_PNARR_CELLTYPE]);
;11130          11221  2            DBG$BUILD_PRIMARY_SUBNODE(.PRIMPTR, RST$K_DATA, 0,
;11131          11222  2                                     .FCODE, .NODEPTR[DBG$L_PNARR_CELLTYPE], 0);
;11132          11223  2            RETURN;
;11133          11224  2
;11134          11225  1            END;


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

5F 54 45 47 5C 52 45 53 52 41 50 47 42 44 1A 03398 P.AYL:   .ASCII  <26>\DBGPARSER\<92>\GET_SUBSCRIPTS 5\      ;
      35 20 53 54 50 49 52 43 53 42 55 53 033A7
5F 54 45 47 5C 52 45 53 52 41 50 47 42 44 1B 033B3 P.AYM:   .ASCII  <27>\DBGPARSER\<92>\GET_SUBSCRIPTS 10\     ;
   30 31 20 53 54 50 49 52 43 53 42 55 53 033C2


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

                                        0FFC 00000 GET_SUBSCRIPTS:
                                                        .WORD   Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11        ; 1065
                                    5E      C0  AE  9E 00002        MOVAB   -64(SP), SP                     ;
                                    52      04  AC  D0 00006        MOVL    PRIMPTR, R2                     ; 1071
                          00000000G 00          52  D0 0000A        MOVL    R2, DBG$GL_CURRENT_PRIMARY      ;
                                 79 8F      02  A2  91 00011        CMPB    2(R2), #121                     ; 1072
                                                0D  13 00016        BEQL    1$                              ;
                                    000287E8    8F  DD 00018        PUSHL   #165864                         ; 1072
                          00000000G 00          01  FB 0001E        CALLS   #1, LIB$SIGNAL                  ;
                                    53      04  A2  9E 00025 1$:    MOVAB   4(R2), R3                       ; 1072
                                    01      02  A3  91 00029        CMPB    2(R3), #1                       ;
                                                03  12 0002D        BNEQ    2$                              ;
                                                0129 31 0002F       BRW     14$                             ;
                                    02      02  A3  91 00032 2$:    CMPB    2(R3), #2                       ; 1073
                                                50  12 00036        BNEQ    4$                              ;
                                                2C  AE  9F 00038    PUSHAB  BITSIZE                         ; 1074
                                                34  AE  9F 0003B    PUSHAB  TYPECODE                        ;
                                                08  A2  DD 0003E    PUSHL   8(R2)                           ;
                                00000000G       00      03  FB 00041    CALLS   #3, DBG$STA_TYP_ATOMIC      ;
                                                18  30  AE  D1 00048    CMPL    TYPECODE, #24               ; 1074
                                                    20  12 0004C        BNEQ    3$                          ;
                                                1C  AE  9F 0004E    PUSHAB  VALPTR                          ; 1074
                                                7E  7A  8F  9A 00051    MOVZBL  #122, -(SP)                 ;
                                                    52  DD 00055        PUSHL   R2                          ;
                                00000000G       00      03  FB 00057    CALLS   #3, DBG$PRIM_TO_VAL         ;
                                20 AE       1C  AE      14  C1 0005E    ADDL3   #20, VALPTR, DSCADDR        ; 1075
                                                50  20  AE  D0 00064    MOVL    DSCADDR, R0                 ; 1075
                                                0E  02  A0  91 00068    CMPB    2(R0), #14                  ;
                                                    37  13 0006C        BEQL    5$                          ;
                                                0E  30  AE  D1 0006E 3$:    CMPL    TYPECODE, #14           ; 1075
                                                    14  12 00072        BNEQ    4$                          ;
                                                36  AE  010E 8F  B0 00074    MOVW    #270, DESCR+2           ; 1076
                                            50  2C  AE      08  C7 0007A    DIVL3   #8, BITSIZE, R0         ; 1076
                                                34  AE      50  B0 0007F    MOVW    R0, DESCR               ;
                                                        34  AE  9F 00083    PUSHAB  DESCR                   ; 1076
                                                            1F  11 00086    BRB     6$                      ;
                                                03      02  A3  91 00088 4$:    CMPB    2(R3), #3           ; 1077
```

```
                                        21   12 0008C          BNEQ    7$                                          ;
                                    20  AE   9F 0008E          PUSHAB  DSCADDR                                     ; 1077
                                    08  A2   DD 00091          PUSHL   @(R2)                                       ;
                        00000000G   00      02   FB 00094    CALLS   #2, DBG$STA_TYP_DESCR                       ;
                                    50  20   AE   D0 0009B    MOVL    DSCADDR, R0                                 ; 1077
                                    0E  02   A0   91 0009F    CMPB    2(R0), #14                                  ;
                                             0A   12 000A3    BNEQ    7$                                          ;
                                             50   DD 000A5 5$: PUSHL   R0                                          ; 1077
                                             52   DD 000A7 6$: PUSHL   R2                                          ;
                              0000V     CF       02   FB 000A9    CALLS   #2, GET_SUBSTRING                       ;
                                             04   00 000AE    RET                                                 ; 1077
                                    06  02   A3   91 000AF 7$: CMPB    2(R3), #6                                   ; 1079
                                             03   13 000B3    BEQL    8$                                          ;
                                          0096    31 000B5    BRW     13$                                         ;
                                00  00000000G   00   8F 000B8 8$: CASEB   DBG$GB_LANGUAGE, #0, #10                 ; 1079
                        008E            008E         008E 000C0 9$: .WORD   13$-9$,-                                ;
                        002D            008E         008E 000C8     13$-9$,-                                        ;
                        008E            0079         008E 000D0     13$-9$,-                                        ;
                                                                    13$-9$,-                                        ;
                                                                    13$-9$,-                                        ;
                                                                    13$-9$,-                                        ;
                                                                    13$-9$,-                                        ;
                                                                    10$-9$,-                                        ;
                                                                    13$-9$,-                                        ;
                                                                    11$-9$,-                                        ;
                                                                    13$-9$                                          ;
                                00000000'   EF   9F 000D6    PUSHAB  P.AYL                                       ; 1086
                                            01   DD 000DC    PUSHL   #1                                          ;
                                00028362    8F   DD 000DE    PUSHL   #164706                                     ;
                        00000000G   00      03   FB 000E4    CALLS   #3, LIB$SIGNAL                              ;
                                            6E   11 000EB    BRB     14$                                         ;
                                    24  AE   9F 000ED 10$:   PUSHAB  VALPTR                                      ; 1081
                                7E  7A  8F   9A 000F0        MOVZBL  #122, -(SP)                                 ;
                                            52   DD 000F4    PUSHL   R2                                          ;
                        00000000G   00      03   FB 000F6    CALLS   #3, DBG$PRIM_TO_VAL                         ;
                                    50  24   AE   D0 000FD    MOVL    VALPTR, R0                                  ; 1081
                                    53  20   A0   D0 00101    MOVL    32(R0), ADDRESS                             ;
                                    56  18   A2   D0 00105    MOVL    24(R2), NODEPTR                             ; 1082
                                    50       66   0F 00109    REMQUE  (NODEPTR), DUMMY                            ; 1082
                                             53   DD 0010C    PUSHL   ADDRESS                                     ; 1082
                                    52  04   AC   D0 0010E    MOVL    PRIMPTR, R2                                 ; 1082
                                        08   A2   DD 00112    PUSHL   8(R2)                                       ;
                        00000000G   00      02   FB 00115    CALLS   #2, DBG$TYPEID_FOR_ARRAY                    ;
                                    58           50   D0 0011C    MOVL    R0, TYPEID                             ;
                                                 7E   D4 0011F    CLRL    -(SP)                                  ; 1082
                                                 58   DD 00121    PUSHL   TYPEID                                 ; 1082
                                                 01   DD 00123    PUSHL   #1                                     ; 1082
                                7E                   06   7D 00125    MOVQ    #6, -(SP)                          ;
                                                     52   DD 00128    PUSHL   R2                                 ;
                            CC1A    CF               06   FB 0012A    CALLS   #6, DBG$BUILD_PRIMARY_SUBNODE      ;
                                    56  18           A2   D0 0012F    MOVL    24(R2), NODEPTR                    ; 1082
                                14  A6               53   D0 00133    MOVL    ADDRESS, 20(NODEPTR)               ; 1082
                                                     22   11 00137    BRB     14$                                ; 1079
                                    06  02           A3   91 00139 11$: CMPB    2(R3), #6                          ; 1084
                                                     09   12 0013D    BNEQ    12$                                ;
                                                     52   DD 0013F    PUSHL   R2                                 ; 1084
                            F91E    CF               01   FB 00141    CALLS   #1, GET_DEREFERENCE                ;
```

```
                                        F1  11 00146      BRB     11$                                       ;
                        01          02  A3  91 00148 12$: CMPB    2(R3), #1                                 ; 1084
                                        0D  13 0014C      BEQL    14$                                       ;
                           000287E8     8F  DD 0014E 13$: PUSHL   #165864                                   ; 1087
              00000000G  00             01  FB 00154      CALLS   #1, LIB$SIGNAL                            ;
          50         04  AC             18  C1 0015B 14$: ADDL3   #24, PRIMPTR, R0                          ; 1088
                         56             60  D0 00160      MOVL    (R0), NODEPTR                             ;
                         52         28  A6  9E 00163      MOVAB   40(R6), SUBVECTOR                         ; 1088
                         5A         08  A6  9E 00167      MOVAB   8(NODEPTR), R10                           ; 1088
                         01         01  AA  91 0016B      CMPB    1(R10), #1                                ;
                                        15  13 0016F      BEQL    15$                                       ;
                           00000000'    EF  9F 00171      PUSHAB  P.AYM                                     ; 1088
                                        01  DD 00177      PUSHL   #1                                        ;
                           00028362     8F  DD 00179      PUSHL   #164706                                   ;
              00000000G  00             03  FB 0017F      CALLS   #3, LIB$SIGNAL                            ;
                                    10  AE  D4 00186 15$: CLRL    THIS_SUBSCR_IS_RANGE                      ; 1089
                         59         1F  A6  9A 00189      MOVZBL  31(NODEPTR), SUBSCR_COUNT                 ; 1089
              00000000'  EF             01  D0 0018D      MOVL    #1, TERMINATOR_CODE                       ; 1089
                         02 00000000'   EF  D1 00194 16$: CMPL    TERMINATOR_CODE, #2                       ; 1089
                                        03  12 0019B      BNEQ    17$                                       ;
                                      030B  31 0019D      BRW     51$                                       ;
 59       1B    A6              08      00  ED 001A0 17$: CMPZV   #0, #8, 27(NODEPTR), SUBSCR_COUNT         ; 1090
                                        13  14 001A6      BGTR    18$                                       ;
                         7E         1B  A6  9A 001A8      MOVZBL  27(NODEPTR), -(SP)                        ; 1090
                                        01  DD 001AC      PUSHL   #1                                        ;
                           00028EB0     8F  DD 001AE      PUSHL   #167600                                   ;
              00000000G  00             03  FB 001B4      CALLS   #3, LIB$SIGNAL                            ;
                     04  AE 00000000'   EF  D0 001BB 18$: MOVL    CHARPTR, LA_PTR                           ; 1091
                         20         04  BE  91 001C3 19$: CMPB    @LA_PTR, #32                              ; 1091
                                        05  12 001C7      BNEQ    20$                                       ;
                                    04  AE  D6 001C9      INCL    LA_PTR                                    ;
                                        F5  11 001CC      BRB     19$                                       ;
                         2A         04  BE  91 001CE 20$: CMPB    @LA_PTR, #42                              ; 1091
                                        03  13 001D2      BEQL    21$                                       ;
                                      00AF  31 001D4      BRW     28$                                       ;
   00000000'    EF      04      AE      01  C1 001D7 21$: ADDL3   #1, LA_PTR, CHARPTR                       ; 1091
                                        7E  D4 001E0      CLRL    -(SP)                                     ; 1092
                           00000000'    EF  DD 001E2      PUSHL   SUBSCRIPT_TERM_TBL                        ; 1092
                                        7E  7C 001E8      CLRQ    -(SP)                                     ; 1092
                   D58C  CF             04  FB 001EA      CALLS   #4, DBG$LEXICAL_SCANNER                   ;
                     18  AE             50  D0 001EF      MOVL    R0, TOKEN                                 ;
                         50 00000000'   EF  9E 001F3      MOVAB   TERMINATOR_TOKEN, R0                      ; 1093
                         50         18  AE  D1 001FA      CMPL    TOKEN, R0                                 ;
                                        1E  13 001FE      BEQL    22$                                       ;
                     28  AE             01  90 00200      MOVB    #1, ASCIC_STRING                          ; 1093
                     29  AE 00000000'   FF  90 00204      MOVB    @CHARPTR, ASCIC_STRING+1                  ; 1093
                                    28  AE  9F 0020C      PUSHAB  ASCIC_STRING                              ; 1093
                                        01  DD 0020F      PUSHL   #1                                        ;
                           000289E2     8F  DD 00211      PUSHL   #166370                                   ;
              00000000G  00             03  FB 00217      CALLS   #3, LIB$SIGNAL                            ;
                         03 00000000'   EF  D1 0021E 22$: CMPL    TERMINATOR_CODE, #3                       ; 1093
                                        0D  12 00225      BNEQ    23$                                       ;
                           00028F08     8F  DD 00227      PUSHL   #167688                                   ; 1094
              00000000G  00             01  FB 0022D      CALLS   #1, LIB$SIGNAL                            ;
                           00000000'    EF  D5 00234 23$: TSTL    TERMINATOR_CODE                           ; 1094
                                        0D  12 0023A      BNEQ    24$                                       ;
                           00028E90     8F  DD 0023C      PUSHL   #167568                                   ; 1094
```

```
                00000000G   00              01   FB 00242              CALLS   #1, LIB$SIGNAL                                  ;
                00000000'   EF 00000000'    EF   C0 00249 24$:        ADDL2   TERMINATOR_LENGTH, CHARPTR                      ; 1094
            25              6A              13   E0 00254              BBS     #19, (R10), 27$                                 ; 1095
                    02      AA              08   88 00258              BISB2   #8, 2(R10)                                     ; 1095
                            51              01   CE 0025C              MNEGL   #1, I                                           ; 1095
                                            18   11 0025F              BRB     26$                                             ;
            50              51              14   C5 00261 25$:         MULL3   #20, I, R0                                      ;
                                    08 A042 9F   00265                 PUSHAB  8(R0)[SUBVECTOR]                                ; 1095
                                       6042 9F   00269                 PUSHAB  (R0)[SUBVECTOR]                                 ;
                            9E              9E   D0 0026C              MOVL    @(SP)+, @(SP)+                                  ;
                                    0C A042 9F   0026F                 PUSHAB  12(R0)[SUBVECTOR]                               ; 1096
                                       6042 9F   00273                 PUSHAB  (R0)[SUBVECTOR]                                 ;
                            9E              9E   D0 00276              MOVL    @(SP)+, @(SP)+                                  ;
            E4              51              59   F2 00279 26$:         AOBLSS  SUBSCR_COUNT, I, 25$                            ; 1095
                                            59   D6 0027D 27$:         INCL    SUBSCR_COUNT                                    ; 1096
                    02      AA              08   88 0027F              BISB2   #8, 2(R10)                                      ; 1096
                                        FF0E 31   00283                 BRW     16$                                             ; 1091
                    14      AE 00000000'    EF   D0 00286 28$:         MOVL    EXPRESSION_RADIX, SAVED_RADIX                   ; 1097
                00000000'   EF              0A   D0 0028E              MOVL    #10, EXPRESSION_RADIX                           ; 1097
                                00000000'   EF   DD 00295              PUSHL   SUBSCRIPT_TERM_TBL                              ; 1098
                                            7E   D4 0029B              CLRL    -(SP)                                           ;
                        D033    CF              02   FB 0029D              CALLS   #2, DBG$EXPRESSION_PARSER                       ;
                          1C    AE              50   D0 002A2              MOVL    R0, VALPTR                                      ;
                00000000'   EF          14      AE   D0 002A6              MOVL    SAVED_RADIX, EXPRESSION_RADIX                   ; 1098
                                0v000000'   EF   D5 002AE              TSTL    TERMINATOR_CODE                                 ; 1098
                                            0D   12 002B4              BNEQ    29$                                             ;
                                00028E90    8F   DD 002B6              PUSHL   #167568                                         ;
                00000000G   00              01   FB 002BC              CALLS   #1, LIB$SIGNAL                                  ;
                00000000'   EF 00000000'    EF   C0 002C3 29$:        ADDL2   TERMINATOR_LENGTH, CHARPTR                      ; 1098
                                            04   DD 002CE              PUSHL   #4                                              ; 1099
                            7E      7A      8F   9A 002D0              MOVZBL  #122, -(SP)                                     ;
                00000000G   00              02   FB 002D4              CALLS   #2, DBG$MAKE_SKELETON_DESC                      ;
                            54              50   D0 002DB              MOVL    R0, DECLTYPE                                    ;
                    07      A4              06   90 002DE              MOVB    #6, 7(DECLTYPE)                                 ; 1099
            55              59              14   C5 002E2              MULL3   #20, SUBSCR_COUNT, R5                           ; 1099
                                    10 A542 9F   002E6                 PUSHAB  16(R5)[SUBVECTOR]                               ;
                            58              9E   D0 002EA              MOVL    @(SP)+, TYPEID                                  ;
                    08      A4              58   D0 002ED              MOVL    TYPEID, 8(DECLTYPE)                             ; 1100
                    18      A4      20      A4   9E 002F1              MOVAB   32(R4), 24(DECLTYPE)                            ; 1100
                            53      14      A4   9E 002F6              MOVAB   20(DECLTYPE), R3                                ; 1101
                                            58   D5 002FA              TSTL    TYPEID                                          ; 1100
                                            0D   12 002FC              BNEQ    30$                                             ;
                    06      A4                   02 90 002FE          MOVB    #2, 6(DECLTYPE)                                 ; 1101
                            63 01080004          8F D0 00302          MOVL    #17301508, (R3)                                 ; 1101
                                                 6A 11 00309          BRB     33$                                             ; 1100
                                        2C      AE   9F 0030B 30$:     PUSHAB  BITSIZE                                         ; 1102
                                            58   DD 0030E              PUSHL   TYPEID                                          ;
                00000000G   00              02   FB 00310              CALLS   #2, DBG$STA_SYMSIZE                             ;
                            5B      18      A8   9A 00317              MOVZBL  24(TYPEID), FCODE                               ; 1102
                    06      A4              5B   90 0031B              MOVB    FCODE, 6(DECLTYPE)                              ; 1102
                            02              5B   D1 0031F              CMPL    FCODE, #2                                       ; 1103
                                            53   12 00322              BNEQ    34$                                             ;
                                        2C      AE   9F 00324          PUSHAB  BITSIZE                                         ; 1103
                                        34      AE   9F 00327          PUSHAB  TYPECODE                                        ;
                                            58   DD 0032A              PUSHL   TYPEID                                          ;
                00000000G   00              03   FB 0032C              CALLS   #3, DBG$STA_TYP_ATOMIC                          ;
                0000009E    8F          30      AE   D1 00333          CMPL    TYPECODE, #T58                                  ; 1103
```

```
                                04  12 0033B          BNEQ    31$                                    ;
              30   AE           28  D0 0033D          MOVL    #40, TYPECODE                          ; 1104
                                7E  D4 00341  31$:    CLRL    -(SP)                                  ; 1104
              57         34     AE  D0 00343          MOVL    TYPECODE, R7                           ; 1104
                                57  DD 00347          PUSHL   R7                                     ;
       00000000G   00           02  FB 00349          CALLS   #2, DBG$MAP_DTYPE_CLASS                ;
              03   A3           50  90 00350          MOVB    R0, 3(R3)                              ;
              02   A3           57  90 00354          MOVB    R7, 2(R3)                              ; 1104
                   01           57  D1 00358          CMPL    R7, #1                                 ; 1104
                                14  13 0035B          BEQL    32$                                    ;
                   29           57  D1 0035D          CMPL    R7, #41                                ; 1104
                                0F  13 00360          BEQL    32$                                    ;
                   22           57  D1 00362          CMPL    R7, #34                                ; 1104
                                0A  13 00365          BEQL    32$                                    ;
                   2A           57  D1 00367          CMPL    R7, #42                                ; 1104
                                05  13 0036A          BEQL    32$                                    ;
                   28           57  D1 0036C          CMPL    R7, #40                                ; 1104
                                26  12 0036F          BNEQ    36$                                    ;
                   63    2C     AE  B0 00371  32$:    MOVW    BITSIZE, (R3)                          ; 1105
                                2C  11 00375  33$:    BRB     37$                                    ; 1103
                   03           5B  D1 00377  34$:    CMPL    FCODE, #3                              ; 1105
                                18  12 0037A          BNEQ    35$                                    ;
                                53  DD 0037C          PUSHL   R3                                     ; 1106
                                58  DD 0037E          PUSHL   TYPEID                                 ;
       00000000G   00           02  FB 00380          CALLS   #2, DBG$STA_TYP_DESCR                  ;
              9E   8F    02     A3  91 00387          CMPB    2(R3), #158                            ; 1106
                                15  12 0038C          BNEQ    37$                                    ;
              02   A3           28  90 0038E          MOVB    #40, 2(R3)                             ; 1106
                                0F  11 00392          BRB     37$                                    ; 1105
                         02     A3  B4 00394  35$:    CLRW    2(R3)                                  ; 1107
 50       2C       AE           07  C1 00397  36$:    ADDL3   #7, BITSIZE, R0                        ; 1107
 51                50           08  C7 0039C          DIVL3   #8, R0, R1                             ;
                   63           51  B0 003A0          MOVW    R1, (R3)                               ;
                                54  DD 003A3  37$:    PUSHL   DECLTYPE                               ; 1108
                         20     AE  DD 003A5          PUSHL   VALPTR                                 ;
                  00000000'     EF  9F 003A8          PUSHAB  DBG$GL_CONVERT_TOKEN                   ;
       00000000G   00           03  FB 003AE          CALLS   #3, DBG$EVAL_LANG_OPERATOR             ;
              1C   AE           50  D0 003B5          MOVL    R0, VALPTR                             ;
              0C   AE    18     A0  D0 003B9          MOVL    24(R0), VALADDR                        ; 1109
                   09 00000000G 00  91 003BE          CMPB    DBG$GB_LANGUAGE, #9                    ; 1109
                                1E  12 003C5          BNEQ    38$                                    ;
                                58  D5 003C7          TSTL    TYPEID                                 ; 1109
                                1A  13 003C9          BEQL    38$                                    ;
                   04           5B  D1 003CB          CMPL    FCODE, #4                              ; 1109
                                15  12 003CE          BNEQ    38$                                    ;
                   53    0C     BE  D0 003D0          MOVL    @VALADDR, R3                           ; 1109
                                53  DD 003D4          PUSHL   R3                                     ;
                                58  DD 003D6          PUSHL   TYPEID                                 ;
       00000000G   00           02  FB 003D8          CALLS   #2, DBG$ENUM_POS                       ;
              08   AE           50  D0 003DF          MOVL    R0, CHECK_VAL                          ;
                                08  11 003E3          BRB     39$                                    ;
                   53    0C     BE  D0 003E5  38$:    MOVL    @VALADDR, R3                           ; 1109
              08   AE           53  D0 003E9          MOVL    R3, CHECK_VAL                          ;
                         08   A542  9F 003ED  39$:    PUSHAB  8(R5)[SUBVECTOR]                       ; 1110
                   9E    0C     AE  D1 003F1          CMPL    CHECK_VAL, @(SP)+                      ;
                                0A  19 003F5          BLSS    40$                                    ;
                         0C   A542  9F 003F7          PUSHAB  12(R5)[SUBVECTOR]                      ; 1110
```

```
                9E      0C    AE   D1 003FB          CMPL    CHECK_VAL, @(SP)+
                              21   15 003FF          BLEQ    41$
                        0C A542    9F 00401 40$:     PUSHAB  12(R5)[SUBVECTOR]                       ; 1110
                           9E      DD 00405          PUSHL   @(SP)+
                        08 A542    9F 00407          PUSHAB  8(R5)[SUBVECTOR]                        ; 1110
                           9E      DD 0040B          PUSHL   @(SP)+
                        10    AE   DD 0040D          PUSHL   CHECK_VAL                               ; 1110
                        01    A9   9F 00410          PUSHAB  1(SUBSCR_COUNT)
                              04   DD 00413          PUSHL   #4
                    0002868B    8F   DD 00415          PUSHL   #165515
            00000000G   00            FB 00418          CALLS   #6, LIB$SIGNAL
                        03 00000000'  EF   D1 00422 41$:     CMPL    TERMINATOR_CODE, #3             ; 1111
                              43   12 00429          BNEQ    45$
                0D          10    AE   E9 0042B          BLBC    THIS_SUBSCR_IS_RANGE, 42$       ; 1112
                    00028F08    8F   DD 0042F          PUSHL   #167688
            00000000G   00            01   FB 00435          CALLS   #1, LIB$SIGNAL
                  10    AE            01   D0 0043C 42$:     MOVL    #1, THIS_SUBSCR_IS_RANGE        ; 1112
                        6E            53   D0 00440          MOVL    R3, LOW_RANGE_VAL               ; 1112
          61            6A            13   E0 00443          BBS     #19, (R10), 50$                 ; 1112
                02      AA            08   88 00447          BISB2   #8, 2(R10)                      ; 1112
                        51            01   CE 0044B          MNEGL   #1, I                           ; 1112
                                      18   11 0044E          BRB     44$
          50            51            14   C5 00450 43$:     MULL3   #20, I, R0                      ; 1113
                        08 A042       9F   00454          PUSHAB  8(R0)[SUBVECTOR]                ; 1113
                           6042       9F   00458          PUSHAB  (R0)[SUBVECTOR]
                9E            9E      D0 0045B          MOVL    @(SP)+, @(SP)+
                        0C A042       9F   0045E          PUSHAB  12(R0)[SUBVECTOR]               ; 1113
                           6042       9F   00462          PUSHAB  (R0)[SUBVECTOR]
                9E            9E      D0 00465          MOVL    @(SP)+, @(SP)+
          E4            51            59   F2 00468 44$:     AOBLSS  SUBSCR_COUNT, I, 43$            ; 1112
                                      3A   11 0046C          BRB     50$                             ; 1111
                        17    10    AE   E9 0046E 45$:     BLBC    THIS_SUBSCR_IS_RANGE, 47$       ; 1115
                        53            6E   D1 00472          CMPL    LOW_RANGE_VAL, R3               ; 1115
                                      0D   15 00475          BLEQ    46$
                    00028F08    8F   DD 00477          PUSHL   #167688
            00000000G   00            01   FB 0047D          CALLS   #1, LIB$SIGNAL
                        10    AE   D4 00484 46$:     CLRL    THIS_SUBSCR_IS_RANGE            ; 1115
                              03   11 00487          BRB     48$                             ; 1115
                        6E            53   D0 00489 47$:     MOVL    R3, LOW_RANGE_VAL               ; 1116
          0F            6A            13   E1 0048C 48$:     BBC     #19, (R10), 49$                 ; 1117
                        08 A542       9F   00490          PUSHAB  8(R5)[SUBVECTOR]                ; 1117
                9E      04    AE      D0 00494          MOVL    LOW_RANGE_VAL, @(SP)+
                        0C A542       9F   00498          PUSHAB  12(R5)[SUBVECTOR]               ; 1117
                9E            53      D0 0049C          MOVL    R3, @(SP)+
                           6542       9F   0049F 49$:     PUSHAB  (R5)[SUBVECTOR]                 ; 1118
                9E      04    AE      D0 004A2          MOVL    LOW_RANGE_VAL, @(SP)+
                              59   D6 004A6          INCL    SUBSCR_COUNT                    ; 1118
                        FCE9          31   004A8 50$:     BRW     16$                             ; 1089
                  1F    A6            59   90 004AB 51$:     MOVB    SUBSCR_COUNT, 31(NODEPTR)       ; 1119
    59    1B    A6            08      00   ED 004AF          CMPZV   #0, #8, 27(NODEPTR), SUBSCR_COUNT ; 1119
                                      1A   13 004B5          BEQL    52$
                        3A 00000000'  EF   E8 004B7          BLBS    MULTIPLE_SUBSCR, 53$            ; 1120
                        7E      1B    A6   9A 004BE          MOVZBL  27(NODEPTR), -(SP)              ; 1120
                                      01   DD 004C2          PUSHL   #1
                    00028EA0    8F   DD 004C4          PUSHL   #167584
            00000000G   00            03   FB 004CA          CALLS   #3, LIB$SIGNAL
                23            6A            13   E0 004D1 52$:     BBS     #19, (R10), 53$         ; 1121
```

```
                      02  AA              01  88 004D5          BISB2   #1, 2(R10)                                  ; 1121
                                       24  A6  DD 004D9          PUSHL   36(NODEPTR)                                 ; 1122
                000000000G  00           01  FB 004DC          CALLS   #1, DBG$STA_TYPEFCODE
                            5B           50  D0 004E3          MOVL    R0, FCODE
                                         7E  D4 004E6          CLRL    -(SP)                                       ; 1122
                                       24  A6  DD 004EB          PUSHL   36(NODEPTR)                                 ; 1122
                                         5B  DD 004EB          PUSHL   FCODE
                                7E           06  7D 004ED          MOVQ    #6, -(SP)                               ; 1122
                                       04  AC  DD 004F0          PUSHL   PRIMPTR
                          C851  CF           06  FB 004F3          CALLS   #6, DBG$BUILD_PRIMARY_SUBNODE
                                             04  004F8 53$:       RET                                             ; 1122

; Routine Size:  1273 bytes,    Routine Base:  DBG$CODE + 3323
```

```
;11136          11226  1        ROUTINE GET_SUBSTRING (PRIMPTR, DSCADDR) : NOVALUE =
;11137          11227  1
;11138          11228  1        !
;11139          11229  1        ! FUNCTION
;11140          11230  1        !       This routine picks up a substring reference.
;11141          11231  1        !       For example, in FORTRAN, if a variable X is declared CHARACTER*n
;11142          11232  1        !       then we want to allow X(i:j), where i and j represent the
;11143          11233  1        !       beginning and ending character positions.
;11144          11234  1        !       In PASCAL, if a variable X is declared PACKED ARRAY[1..N] OF CHAR
;11145          11235  1        !       then we want to allow subscripting X[i] or ranged subscripting
;11146          11236  1        !       X[i:j] to get at substrings of X.
;11147          11237  1        !
;11148          11238  1        !       This routine gets called from the GET_SUBSCRIPTS routine, at
;11149          11239  1        !       the point where we discover that what we have is not an
;11150          11240  1        !       array, but is a string.
;11151          11241  1        !
;11152          11242  1        !       The expression parser is called to pick up the first subscript.
;11153          11243  1        !       If the terminator ":" is encountered then the expression parser
;11154          11244  1        !       is called again to pick up the upper bound.
;11155          11245  1        !
;11156          11246  1        !       These substring bounds then get translated into the PRIM_OFFSET
;11157          11247  1        !       and PRIM_LENGTH fields of the Primary Descriptor, and the
;11158          11248  1        !       SUBREF flag is lit to indicate that a substring selection has
;11159          11249  1        !       taken place.
;11160          11250  1        !
;11161          11251  1        ! INPUTS
;11162          11252  1        !       PRIMPTR - A pointer to the Primary Descriptor for a string
;11163          11253  1        !                 about to be subscripted.
;11164          11254  1        !       DSCADDR - A pointer to the string descriptor representing
;11165          11255  1        !                 the string to be subscripted.
;11166          11256  1        !
;11167          11257  1        ! OUTPUTS
;11168          11258  1        !       The PRIMPTR Primary Descriptor is changed to include the
;11169          11259  1        !       substring information.
;11170          11260  2          BEGIN
;11171          11261  2
;11172          11262  2          MAP
;11173          11263  2              PRIMPTR: REF DBG$PRIMARY,
;11174          11264  2              DSCADDR: REF DBG$STG_DESC;
;11175          11265  2
;11176          11266  2          LOCAL
;11177          11267  2              HIGH_VALUE,                     ! Subscript value
;11178          11268  2              LOW_VALUE,                      ! Subscript value
;11179          11269  2              NODEPTR: REF DBG$PRIM_NODE,     ! Pointer to Primary Subnode
;11180          11270  2              SAVED_RADIX,                    ! Temporary to save radix
;11181          11271  2              VALPTR: REF DBG$VALDESC;        ! Pointer to a Value Descriptor
;11182          11272  2
;11183          11273  2
;11184          11274  2
;11185          11275  2          ! Temporarily set the radix to decimal. Then call the Expression Parser
;11186          11276  2          ! to pick up the lower string bound.
;11187          11277  2          !
;11188          11278  2          SAVED_RADIX = .EXPRESSION_RADIX;
;11189          11279  2          EXPRESSION_RADIX = DBG$K_DECIMAL;
;11190          11280  2          VALPTR = DBG$EXPRESSION_PARSER (FALSE, .SUBSCRIPT_TERM_TBL);
;11191          11281  2          EXPRESSION_RADIX = .SAVED_RADIX;
;11192          11282  2
```

```
;11193          11283  2
;11194          11284  2          ! Check the terminator code. If there was no terminator, (the
;11195          11285  2          ! input line just ended), signal an error. If the terminator was
;11196          11286  2          ! a comma, this is also an error - we only allow the colon or
;11197          11287  2          ! the closing subscript in this case.
;11198          11288  2          !
;11199          11289  2          IF .TERMINATOR_CODE EQL TOKEN$K_TERM_NONE
;11200          11290  2          THEN
;11201          11291  2              SIGNAL (DBG$_MISCLOSUB);
;11202          11292  2          IF .TERMINATOR_CODE EQL TOKEN$K_TERM_COMMA
;11203          11293  2          THEN
;11204          11294  2              SIGNAL (DBG$_SYNERREXPR, 1, UPLIT BYTE (%ASCIC ','));
;11205          11295  2          CHARPTR = .CHARPTR + .TERMINATOR_LENGTH;
;11206          11296  2
;11207          11297  2          ! Convert the subscript value to longword integer.
;11208          11298  2          !
;11209          11299  2          LOW_VALUE = CONVERT_TO_INTEGER (.VALPTR);
;11210          11300  2
;11211          11301  2          ! Check for ":" terminator. This indicates we also have to pick
;11212          11302  2          ! up the high value.
;11213          11303  2          !
;11214          11304  2          IF .TERMINATOR_CODE EQL TOKEN$K_TERM_COLON
;11215          11305  2          THEN
;11216          11306  3              BEGIN
;11217          11307  3
;11218          11308  3              ! Pick up another expression for the high value.
;11219          11309  3              !
;11220          11310  3              SAVED_RADIX = .EXPRESSION_RADIX;
;11221          11311  3              EXPRESSION_RADIX = DBG$K_DECIMAL;
;11222          11312  3              VALPTR = DBG$EXPRESSION_PARSER (FALSE, .SUBSCRIPT_TERM_TBL);
;11223          11313  3              EXPRESSION_RADIX = .SAVED_RADIX;
;11224          11314  3
;11225          11315  3              ! Check for any of end-of-line, comma, or colon. These are all
;11226          11316  3              ! errors here.
;11227          11317  3              !
;11228          11318  3              IF .TERMINATOR_CODE EQL TOKEN$K_TERM_NONE
;11229          11319  3              THEN
;11230          11320  3                  SIGNAL (DBG$_MISCLOSUB);
;11231          11321  3              IF .TERMINATOR_CODE EQL TOKEN$K_TERM_COMMA
;11232          11322  3              THEN
;11233          11323  3                  SIGNAL (DBG$_SYNERREXPR, 1, UPLIT BYTE (%ASCIC ','));
;11234          11324  3              IF .TERMINATOR_CODE EQL TOKEN$K_TERM_COMMA
;11235          11325  3              THEN
;11236          11326  3                  SIGNAL (DBG$_SYNERREXPR, 1, UPLIT BYTE (%ASCIC ':'));
;11237          11327  3              CHARPTR = .CHARPTR + .TERMINATOR_LENGTH;
;11238          11328  3
;11239          11329  3              ! Convert the value descriptor to an integer.
;11240          11330  3              !
;11241          11331  3              HIGH_VALUE = CONVERT_TO_INTEGER (.VALPTR);
;11242          11332  3              END
;11243          11333  3
;11244          11334  3          ! No high value present - same as low value.
;11245          11335  3          !
;11246          11336  2          ELSE
;11247          11337  2              HIGH_VALUE = .LOW_VALUE;
;11248          11338  2
;11249          11339  2          ! Signal an error if the range is reversed.
```

```
:11250          11340  2          !
:11251          11341  2          IF .LOW_VALUE LSS 1
:11252          11342  2          OR .HIGH_VALUE GTR .DSCADDR[DSC$W_LENGTH]
:11253          11343  2          OR .LOW_VALUE GTR .HIGH_VALUE
:11254          11344  2          THEN
:11255          11345  2              SIGNAL (DBG$_SUBSTRING, 3, .LOW_VALUE, .HIGH_VALUE,
:11256          11346  2                      .DSCADDR[DSC$W_LENGTH]);
:11257          11347  2
:11258          11348  2          ! Signal an error if the values are too large to fit in a Primary.
:11259          11349  2          !
:11260          11350  2          IF .LOW_VALUE GTR %X'7FFF'
:11261          11351  2          THEN
:11262          11352  2              SIGNAL(DBG$_ILLOFFSET, 1, .LOW_VALUE);
:11263          11353  2          IF (1+.HIGH_VALUE-.LOW_VALUE) GTR %X'7FFF'
:11264          11354  2          THEN
:11265          11355  2              SIGNAL(DBG$_ILLSUBLEN);
:11266          11356  2
:11267          11357  2          ! Modify the primary to indicate the substring information
:11268          11358  2          !
:11269          11359  2          NODEPTR = .PRIMPTR [DBG$L_PRIM_BLINK];
:11270          11360  2          PRIMPTR [DBG$V_DHDR_SUBREF] = TRUE;
:11271          11361  2          PRIMPTR [DBG$W_PRIM_OFFSET] = .LOW_VALUE;
:11272          11362  2          PRIMPTR [DBG$W_PRIM_LENGTH] = 1 + .HIGH_VALUE - .LOW_VALUE;
:11273          11363  2          NODEPTR [DBG$L_PNODE_RELOC] = -1;
:11274          11364  1          END;


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

                                2C  01  033CF P.AYN:    .ASCII  <1>\,\                          ;
                                2C  01  033D1 P.AYO:    .ASCII  <1>\,\                          ;
                                3A  01  033D3 P.AYP:    .ASCII  <1>\:\                          ;


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

                                    00FC 00000 GET_SUBSTRING:
                                                .WORD   Save R2,R3,R4,R5,R6,R7                  ; 1122
                            57 00000000'    EF  9E 00002        MOVAB   P.AYN, R7               ;
                            56 00000000G    00  9E 00009        MOVAB   LIB$SIGNAL, R6          ;
                            55 00000000'    EF  9E 00010        MOVAB   EXPRESSION_RADIX, R5    ;
                            52              65  D0 00017        MOVL    EXPRESSION_RADIX, SAVED_RADIX   ; 1127
                            65              0A  D0 0001A        MOVL    #10, EXPRESSION_RADIX   ; 1127
                                    20      A5  DD 0001D        PUSHL   SUBSCRIPT_TERM_TBL      ; 1128
                                            7E  D4 00020        CLRL    -(SP)                   ;
                        CDB5    CF          02  FB 00022        CALLS   #2, DBG$EXPRESSION_PARSER   ;
                            54              50  D0 00027        MOVL    R0, VALPTR              ;
                            65              52  D0 0002A        MOVL    SAVED_RADIX, EXPRESSION_RADIX   ; 1128
                                    28      A5  D5 0002D        TSTL    TERMINATOR_CODE         ; 1128
                                            09  12 00030        BNEQ    1$                      ;
                                00028E90    8F  DD 00032        PUSHL   #167568                 ; 1129
                            66              01  FB 00038        CALLS   #1, LIB$SIGNAL          ;
                            01      28      A5  D1 0003B 1$:    CMPL    TERMINATOR_CODE, #1     ; 1129
                                            0D  12 0003F        BNEQ    2$                      ;
                                            57  DD 00041        PUSHL   R7                      ; 1129
```

```
                           01 DD 00043          PUSHL   #1                                    :
              000289E2     8F DD 00045          PUSHL   #166370                               :
            66             03 FB 0004B          CALLS   #3, LIB$SIGNAL                        :
      FBF4  C5       2C    A5 C0 0004E 2$:      ADDL2   TERMINATOR_LENGTH, CHARPTR            : 1129
                           54 DD 00054          PUSHL   VALPTR                                : 1129
      EC26  CF             01 FB 00056          CALLS   #1, CONVERT_TO_INTEGER                :
            53             50 D0 0005B          MOVL    R0, LOW_VALUE                         :
            03       28    A5 D1 0005E          CMPL    TERMINATOR_CODE, #3                   : 1130
                           5E 12 00062          BNEQ    6$                                    :
            52             65 D0 00064          MOVL    EXPRESSION_RADIX, SAVED_RADIX         : 1131
            65             0A D0 00067          MOVL    #10, EXPRESSION_RADIX                 : 1131
                     20    A5 DD 0006A          PUSHL   SUBSCRIPT_TERM_TBL                    : 1131
                           7E D4 0006D          CLRL    -(SP)                                 :
      CD68  CF             02 FB 0006F          CALLS   #2, DBG$EXPRESSION_PARSER             :
            54             50 D0 00074          MOVL    R0, VALPTR                            :
            65             52 D0 00077          MOVL    SAVED_RADIX, EXPRESSION_RADIX         : 1131
                     28    A5 D5 0007A          TSTL    TERMINATOR_CODE                       : 1131
                           09 12 0007D          BNEQ    3$                                    :
              00028E90     8F DD 0007F          PUSHL   #167568                               : 1132
            66             01 FB 00085          CALLS   #1, LIB$SIGNAL                        :
            01       28    A5 D1 00088 3$:      CMPL    TERMINATOR_CODE, #1                   : 1132
                           0E 12 0008C          BNEQ    4$                                    :
                     02    A7 9F 0008E          PUSHAB  P.AYO                                 : 1132
                           01 DD 00091          PUSHL   #1                                    :
              000289E2     8F DD 00093          PUSHL   #166370                               :
            66             03 FB 00099          CALLS   #3, LIB$SIGNAL                        :
            01       28    A5 D1 0009C 4$:      CMPL    TERMINATOR_CODE, #1                   : 1132
                           0E 12 000A0          BNEQ    5$                                    :
                     04    A7 9F 000A2          PUSHAB  P.AYP                                 : 1132
                           01 DD 000A5          PUSHL   #1                                    :
              000289E2     8F DD 000A7          PUSHL   #166370                               :
            66             03 FB 000AD          CALLS   #3, LIB$SIGNAL                        :
      FBF4  C5       2C    A5 C0 000B0 5$:      ADDL2   TERMINATOR_LENGTH, CHARPTR            : 1132
                           54 DD 000B6          PUSHL   VALPTR                                : 1133
      EBC4  CF             01 FB 000B8          CALLS   #1, CONVERT_TO_INTEGER                :
            52             50 D0 000BD          MOVL    R0, HIGH_VALUE                        :
                           03 11 000C0          BRB     7$                                    : 1130
            52             53 D0 000C2 6$:      MOVL    LOW_VALUE, HIGH_VALUE                 : 1133
                           53 D5 000C5 7$:      TSTL    LOW_VALUE                             : 1134
                           0D 15 000C7          BLEQ    8$                                    :
 52     08   BC            10 00 ED 000C9       CMPZV   #0, #16, @DSCADDR, HIGH_VALUE         : 1134
                           05 19 000CF          BLSS    8$                                    :
            52             53 D1 000D1          CMPL    LOW_VALUE, HIGH_VALUE                 : 1134
                           13 15 000D4          BLEQ    9$                                    :
            7E       08    BC 3C 000D6 8$:      MOVZWL  @DSCADDR, -(SP)                       : 1134
                           52 DD 000DA          PUSHL   HIGH_VALUE                            : 1134
                           53 DD 000DC          PUSHL   LOW_VALUE                             :
                           03 DD 000DE          PUSHL   #3                                    :
              000280D8     8F DD 000E0          PUSHL   #164056                               :
            66             05 FB 000E6          CALLS   #5, LIB$SIGNAL                        :
  00007FFF  8F             53 D1 000E9 9$:      CMPL    LOW_VALUE, #32767                     : 1135
                           0D 15 000F0          BLEQ    10$                                   :
                           53 DD 000F2          PUSHL   LOW_VALUE                             : 1135
                           01 DD 000F4          PUSHL   #1                                    :
              000280F0     8F DD 000F6          PUSHL   #164080                               :
            66             03 FB 000FC          CALLS   #3, LIB$SIGNAL                        :
                           52 D6 000FF 10$:     INCL    R2                                    : 1135
```

```
                        50      7FFF    C3  9E 00101          MOVAB   32767(R3), R0                          ;
                        50              52  D1 00106          CMPL    R2, R0                                 ;
                                        09  15 00109          BLEQ    11$                                    ;
                    000280F8            8F  DD 0010B          PUSHL   #164088                                ; 1135
                        66              01  FB 00111          CALLS   #1, LIB$SIGNAL                         ;
                        50        04    AC  D0 00114 11$:     MOVL    PRIMPTR, R0                            ; 1135
                        51        18    A0  D0 00118          MOVL    24(R0), NODEPTR                        ;
                    04  A0              02  88 0011C          BISB2   #2, 4(R0)                              ; 1136
                    10  A0              53  B0 00120          MOVW    LOW_VALUE, 16(R0)                      ; 1136
                12  A0                  53  A3 00124          SUBW3   LOW_VALUE, R2, 18(R0)                  ; 1136
                    14  A1              01  CE 00129          MNEGL   #1, 20(NODEPTR)                        ; 1136
                                        04  0012D          RET                                            ; 1136

; Routine Size:  302 bytes,    Routine Base:  DBG$CODE + 381C
```

```
;11276      11365  1        ROUTINE OPERATOR_TO_RESTORE_RADIX =
;11277      11366  1        !
;11278      11367  1        ! FUNCTION
;11279      11368  1        !       This routine returns the Operator Lexical Token Entry for the
;11280      11369  1        !       operator which will restore the currently set expression radix.
;11281      11370  1        !       It is used in the processing of the lexical operators %DEC, %HEX,
;11282      11371  1        !       %OCT, and %BIN.  If the current radix is decimal, for example,
;11283      11372  1        !       when the %HEX operator is encountered, then the decimal radix must
;11284      11373  1        !       be restored when the range of the %HEX operator ends.  Consider
;11285      11374  1        !       this example:
;11286      11375  1        !
;11287      11376  1        !                               10 + %HEX (20 + 30) - 40
;11288      11377  1        !
;11289      11378  1        !       Here 10 and 40 are interpreted as decimal numbers while 20 and 30
;11290      11379  1        !       are treated as hexadecimal numbers.  When the %HEX operator is
;11291      11380  1        !       encountered, this routine is called and returns the %DEC operator.
;11292      11381  1        !       The %DEC operator is pushed onto the operator stack and the radix
;11293      11382  1        !       is then set to hexadecimal.  When the minus sign is encountered,
;11294      11383  1        !       it forces evaluation of the stacked %DEC operator which restores
;11295      11384  1        !       the radix to decimal.
;11296      11385  1        !
;11297      11386  1        !       The current radix value is stored in and picked up from the OWN
;11298      11387  1        !       variable EXPRESSION_RADIX.
;11299      11388  1        !
;11300      11389  1        ! INPUTS
;11301      11390  1        !       NONE
;11302      11391  1        !
;11303      11392  1        ! OUTPUTS
;11304      11393  1        !       The return value of this routine is the address of the Operator
;11305      11394  1        !               Lexical Token Entry for the lexical operator which
;11306      11395  1        !               restores the current value of EXPRESSION_RADIX.
;11307      11396  1        !
;11308      11397  1
;11309      11398  2            BEGIN
;11310      11399  2
;11311      11400  2
;11312      11401  2            ! Based on the current radix value, return the lexical operator which
;11313      11402  2            ! will restore that radix value when popped from the operator stack
;11314      11403  2            ! and evaluated.
;11315      11404  2            !
;11316      11405  2            IF .EXPRESSION_RADIX EQL DBG$K_DECIMAL THEN RETURN RADIX_OP_DEC;
;11317      11406  2            IF .EXPRESSION_RADIX EQL DBG$K_HEX     THEN RETURN RADIX_OP_HEX;
;11318      11407  2            IF .EXPRESSION_RADIX EQL DBG$K_OCTAL   THEN RETURN RADIX_OP_OCT;
;11319      11408  2            IF .EXPRESSION_RADIX EQL DBG$K_BINARY  THEN RETURN RADIX_OP_BIN;
;11320      11409  2
;11321      11410  2
;11322      11411  2            ! Any other current radix value is an internal DEBUG error.
;11323      11412  2            !
;11324      11413  2            $DBG_ERROR('DBGPARSER\OPERATOR_TO_RESTORE_RADIX');
;11325      11414  2            RETURN 0;
;11326      11415  2
;11327      11416  1            END;


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

```
52 45 50 4F 5C 52 45 53 52 41 50 47 42 44 23  033D5 P.AYQ:   .ASCII  \#DBGPARSER\<92>\OPERATOR_TO_RESTORE_RAD\    ;
45 52 4F 54 53 45 52 5F 4F 54 5F 52 4F 54 41  033E4                                                            ;
                                              033F3                                                            ;
                                           58 49 033F7          .ASCII  \IX\                                     ;


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

                                     0004 00000 OPERATOR_TO_RESTORE_RADIX:
                                                                .WORD   Save R2                                  ; 1136
                                52 00000000'  EF  9E 00002      MOVAB   RADIX_OP_DEC, R2                        ;
                                51 00000000'  EF  D0 00009      MOVL    EXPRESSION_RADIX, R1                     ; 1140
                                0A            51  D1 00010      CMPL    R1, #10                                  ;
                                              04  12 00013      BNEQ    1$                                       ;
                                50            62  9E 00015      MOVAB   RADIX_OP_DEC, R0                         ;
                                              04  00018      RET                                              ;
                                10            51  D1 00019 1$:  CMPL    R1, #16                                  ; 1140
                                              05  12 0001C      BNEQ    2$                                       ;
                                50         11 A2  9E 0001E      MOVAB   RADIX_OP_HEX, R0                         ;
                                              04  00022      RET                                              ;
                                08            51  D1 00023 2$:  CMPL    R1, #8                                   ; 1140
                                              05  12 00026      BNEQ    3$                                       ;
                                50         22 A2  9E 00028      MOVAB   RADIX_OP_OCT, R0                         ;
                                              04  0002C      RET                                              ;
                                02            51  D1 0002D 3$:  CMPL    R1, #2                                   ; 1140
                                              05  12 00030      BNEQ    4$                                       ;
                                50         33 A2  9E 00032      MOVAB   RADIX_OP_BIN, R0                         ;
                                              04  00036      RET                                              ;
                                      30A9    C2  9F 00037 4$:  PUSHAB  P.AYQ                                    ; 1141
                                              01  DD 0003B      PUSHL   #1                                       ;
                                   00028362   8F  DD 0003D      PUSHL   #164706                                  ;
                          00000000G  00               03  FB 00043      CALLS   #3, LIB$SIGNAL                   ;
                                              50  D4 0004A      CLRL    R0                                       ; 1141
                                              04  0004C      RET                                              ; 1141

; Routine Size:  77 bytes,    Routine Base:  DBG$CODE + 394A
```

```
;11329          11417  1        ROUTINE PATHNAME_TO_PRIMARY(PATHDESC, SUBSCR_DESC, PLIPTR,
;11330          11418  1                                    SAVED_PATHDESC, ARRAY_FLAG) =
;11331          11419  1        !
;11332          11420  1        ! FUNCTION
;11333          11421  1        !       This routine builds a Primary Descriptor Root Node in temporary
;11334          11422  1        !       memory and returns a pointer to that descriptor.  The symbol for
;11335          11423  1        !       which the descriptor is built is identified by a Pathname Descrip-
;11336          11424  1        !       tor.  This routine passes the Pathname Descriptor to the GETSYMBOL
;11337          11425  1        !       routine to get the symbol's SYMID, and then builds the Primary
;11338          11426  1        !       Descriptor from that SYMID.  For data items, the symbol's FCODE
;11339          11427  1        !       and TYPEID (which identify the data type) are also retrieved and
;11340          11428  1        !       added to the Primary Descriptor.
;11341          11429  1        !
;11342          11430  1        !       After the Root Node has been build, DBG$BUILD_PRIMARY_SUBNODE is called
;11343          11431  1        !       to build a Primary Descriptor Sub-Node for the symbol.  This means
;11344          11432  1        !       that the Primary Descriptor is complete when PATHNAME_TO_PRIMARY
;11345          11433  1        !       returns unless further subscripting or other qualification causes
;11346          11434  1        !       additional Sub-Nodes to be appended later.
;11347          11435  1        !
;11348          11436  1        ! INPUTS
;11349          11437  1        !       PATHDESC - The address of a Pathname Descriptor which defines the
;11350          11438  1        !                  symbol for which a Primary Descriptor is to be built.
;11351          11439  1        !       SUBSCR_DESC -   Some languages collect their subscripts as they
;11352          11440  1        !                  pick up the Primary, and do not apply them until the
;11353          11441  1        !                  end. In this case, SUBSCR_DESC is a data structure
;11354          11442  1        !                  containing the saved subscripts.
;11355          11443  1        !       PLIPTR -  In a PL/I exression of the form A->B, we first save away
;11356          11444  1        !                  then Primary for "A". This Primary is in PLIPTR,
;11357          11445  1        !                  and must be appended to the beginning of the Primary
;11358          11446  1        !                  we now build.
;11359          11447  1        !
;11360          11448  1        !       SAVED_PATHDESC: PTH$PATHNAME -
;11361          11449  1        !                  This is an area of storage in which we can save
;11362          11450  1        !                  away a copy of the pathname. This is used to make
;11363          11451  1        !                  pathnames sticky in expressions such as
;11364          11452  1        !                  P1\A->B
;11365          11453  1        !
;11366          11454  1        !       ARRAY_FLAG - An optional fifth parameter, which, if present,
;11367          11455  1        !                  indicates that this routine was called as part of an
;11368          11456  1        !                  array subscripting operation. This is used in BASIC,
;11369          11457  1        !                  where there may be two variables A, one an array and
;11370          11458  1        !                  one not, and it is determined from context which is
;11371          11459  1        !                  meant. This flag is just passed along to GETSYMBOL so
;11372          11460  1        !                  that it can resolve the ambiguity properly.
;11373          11461  1        !
;11374          11462  1        ! OUTPUTS
;11375          11463  1        !       A Primary Descriptor for the symbol specified by PATHDESCR is built
;11376          11464  1        !                  and its address is returned as the routine value.
;11377          11465  1        !
;11378          11466  1
;11379          11467  2            BEGIN
;11380          11468  2
;11381          11469  2            MAP
;11382          11470  2                PATHDESC: REF PTH$PATHNAME,      ! Pointer to input Pathname Descriptor
;11383          11471  2                SAVED_PATHDESC: REF PTH$PATHNAME,! Saved copy of pathname.
;11384          11472  2                SUBSCR_DESC: REF SUBSCR$DESC;    ! Pointer to subscript information
;11385          11473  2
```

```
;11386          11474  2          BUILTIN
;11387          11475  2              ACTUALCOUNT;
;11388          11476  2
;11389          11477  2          LOCAL
;11390          11478  2              ARR_FLAG,                       ! TRUE if expecting an array.
;11391          11479  2              BITSIZE,                        ! Bit length
;11392          11480  2              DESCR: REF DBG$STG_DESC,        ! String Descriptor
;11393          11481  2              DUMMY,                          ! Unused output parameter
;11394          11482  2              DTYPE,                          ! A type code
;11395          11483  2              EXPECTED_SUBS,                  ! Count of expected number of subscripts
;11396          11484  2              FCODE,                          ! The data type format code for symbol
;11397          11485  2              INDEX,                          ! Index into SYMID list
;11398          11486  2              KIND,                           ! RST symbol kind for current symbol
;11399          11487  2              LEN,                            ! Byte length
;11400          11488  2              NODEPTR: REF DBG$PRIM_NODE,     ! Pointer to Primary sub-node
;11401          11489  2              PATHSTRING,                     ! Pointer to pathname Counted ASCII
;11402          11490  2                                              !     string--used for messages
;11403          11491  2              PATHVECTOR1: REF VECTOR[,LONG], ! Pathname vector
;11404          11492  2              PATHVECTOR2: REF VECTOR[,LONG], ! Pathname vector
;11405          11493  2              PICKED_UP_SUBSTRING,            ! flag for when we picked up a substring
;11406          11494  2              PRID: REF PRID$ENTRY,           ! Pointer to Predefined Identifier Entry
;11407          11495  2              PRIMPTR: REF DBG$PRIMARY,       ! Pointer to Primary Descriptor
;11408          11496  2              SCOPE,                          ! Scope where symbol was looked up
;11409          11497  2              SCOPE_STATE,                    ! Kind of scope for SCOPE
;11410          11498  2              SUBSCR_INDEX,                   ! Index into SUBSCR_DESC
;11411          11499  2              SUBVECTOR:                      ! Pointer to subscript vector in
;11412          11500  2                  REF DBG$PRIM_NODE_SUBS,     !     Primary Descriptor Sub-node
;11413          11501  2              SYMID: REF RST$ENTRY,           ! SYMID (Symbol ID) for current symbol
;11414          11502  2              SYMID1: REF RST$ENTRY,          ! scratch SYMID (Symbol ID)
;11415          11503  2              SYMID2: REF RST$ENTRY,          ! scratch SYMID (Symbol ID)
;11416          11504  2              SYMID_VECT: VECTOR[DBG$K_PATHNAME_SIZE], ! Vector of saved SYMIDs
;11417          11505  2              TOOFEWSUB,                      ! Flag saying too few subscripts
;11418          11506  2                                              !     were supplied.
;11419          11507  2              TYPCOMPLST: REF VECTOR[,LONG],
;11420          11508  2              TYPEID: REF RST$ENTRY;          ! The Type ID for the symbol's data type
;11421          11509  2
;11422          11510  2
;11423          11511  2          ! If the PLIPTR field is not zero then we already have part of the
;11424          11512  2          ! Primary.
;11425          11513  2          !
;11426          11514  2          IF .PLIPTR NEQ 0
;11427          11515  2          THEN
;11428          11516  3              BEGIN
;11429          11517  3              PRIMPTR = .PLIPTR;
;11430          11518  3              NODEPTR = .PRIMPTR[DBG$L_PRIM_BLINK];
;11431          11519  3              NODEPTR[DBG$V_PNODE_EVAL] = TRUE;
;11432          11520  3
;11433          11521  3              IF .PATHDESC[PTH$B_PATHCNT] GTR 1
;11434          11522  3              THEN
;11435          11523  3
;11436          11524  3                  ! Save away this pathname.
;11437          11525  3                  !
;11438          11526  3                  CH$MOVE(DBG$K_PATHNAME_SIZE, .PATHDESC, .SAVED_PATHDESC)
;11439          11527  3
;11440          11528  3              ELSE
;11441          11529  3                  IF .SAVED_PATHDESC[PTH$B_PATHCNT] GTR 1
;11442          11530  3                  THEN
```

```
;11443          11531  4                    BEGIN
;11444          11532  4                    LOCAL
;11445          11533  4                        I;
;11446          11534  4
;11447          11535  4                    ! Merge the previous pathname into this one.
;11448          11536  4                    !
;11449          11537  4                    PATHVECTOR1 = PATHDESC[PTH$A_PATHVECTOR];
;11450          11538  4                    PATHVECTOR2 = SAVED_PATHDESC[PTH$A_PATHVECTOR];
;11451          11539  4                    I = .SAVED_PATHDESC[PTH$B_PATHCNT]-1;
;11452          11540  4                    DECR J FROM .PATHDESC[PTH$B_TOTCNT]-1 TO 0 DO
;11453          11541  4                        PATHVECTOR1[.I+.J] = .PATHVECTOR1[.J];
;11454          11542  4                    DECR J FROM .I-1 TO 0 DO
;11455          11543  4                        PATHVECTOR1[.J] = .PATHVECTOR2[.J];
;11456          11544  4                    PATHDESC[PTH$B_TOTCNT] = .PATHDESC[PTH$B_TOTCNT] + .I;
;11457          11545  4                    PATHDESC[PTH$B_PATHCNT] = .PATHDESC[PTH$B_PATHCNT] + .I;
;11458          11546  4                    ! <<< INVOCNUM
;11459          11547  3                    END;
;11460          11548  3                END
;11461          11549  3
;11462          11550  3            ELSE
;11463          11551  3                BEGIN
;11464          11552  3
;11465          11553  3                ! Allocate space for the Primary Descriptor and fill in the descriptor
;11466          11554  3                ! header fields and sub-node links.
;11467          11555  3                !
;11468          11556  3                PRIMPTR = DBG$GET_TEMPMEM(DBG$K_PRIMARY_SIZE);
;11469          11557  3                PRIMPTR[DBG$B_DHDR_TYPE] = DBG$K_PRIMARY_DESC;
;11470          11558  3                PRIMPTR[DBG$B_DHDR_LANG] = .DBG$GB_LANGUAGE;
;11471          11559  3                PRIMPTR[DBG$W_DHDR_LENGTH] = DBG$K_PRIMARY_SIZE+%UPVAL;
;11472          11560  3                PRIMPTR[DBG$L_PRIM_FLINK] = PRIMPTR[DBG$A_PRIM_FLINK];
;11473          11561  3                PRIMPTR[DBG$L_PRIM_BLINK] = PRIMPTR[DBG$A_PRIM_FLINK];
;11474          11562  3
;11475          11563  3                ! Save away copy of first pathname that we see.
;11476          11564  3                !
;11477          11565  3                IF .PATHDESC[PTH$B_PATHCNT] GTR 1
;11478          11566  3                THEN
;11479          11567  3                    CH$MOVE(DBG$K_PATHNAME_SIZE, .PATHDESC, .SAVED_PATHDESC)
;11480          11568  3                ELSE
;11481          11569  3                    SAVED_PATHDESC[PTH$B_PATHCNT] = 0;
;11482          11570  2                END;
;11483          11571  2
;11484          11572  2            DBG$GL_CURRENT_PRIMARY = .PRIMPTR;
;11485          11573  2
;11486          11574  2            ! Call GETSYMBOL to get the KIND and SYMID for the symbol.
;11487          11575  2            ! We pass along the flag saying whether this symbol was seen in
;11488          11576  2            ! a subscripted form - getsymbol can use that to resolve ambiguities
;11489          11577  2            ! in BASIC.
;11490          11578  2            ! This also gives the scope where the symbol
;11491          11579  2            ! was looked up and we save that in the Primary Root Node.
;11492          11580  2            ! Then case on the KIND to decide what to do next.
;11493          11581  2            !
;11494          11582  2            IF .COMPONENTS_IN_PATHNAME
;11495          11583  2            THEN
;11496          11584  2                ARR_FLAG = .SUBSCR_DESC[0, SUBSCR$B_SUBCNT] GTR 0
;11497          11585  2            ELSE
;11498          11586  2                ARR_FLAG = ACTUALCOUNT() GTR 4;
;11499          11587  2            DBG$STA_GETSYMBOL(.PATHDESC, SYMID, KIND, SCOPE_STATE, SCOPE, .ARR_FLAG, FALSE);
```

```
;11500      11588  2          PRIMPTR[DBG$B_PRIM_SCOPE_STATE] = .SCOPE_STATE;
;11501      11589  2          PRIMPTR[DBG$L_PRIM_SCOPE] = .SCOPE;
;11502      11590  2
;11503      11591  2          SELECTONE .KIND OF
;11504      11592  2              SET
;11505      11593  2
;11506      11594  2
;11507      11595  2              ! Handle the case where the symbol is not found in DEBUG's symbol
;11508      11596  2              ! table (i.e., the RST).  Signal the appropriate error message.
;11509      11597  2              !
;11510      11598  2              [RST$K_INVALID]:
;11511      11599  3                  BEGIN
;11512      11600  3
;11513      11601  3                      ! First check for Predefined Identifier reserved by the language.
;11514      11602  3                      ! Note: We do this after we are sure that the symbol is not in
;11515      11603  3                      !       the symbol table.  In PASCAL you may redefine the predefined
;11516      11604  3                      !       symbols.  Currently, PASCAL is the only language where we
;11517      11605  3                      !       have predefined identifiers.  FORTRAN has them but they are
;11518      11606  3                      !       prefaced by a dot "." and are picked up by now.
;11519      11607  3                      ! Check that no invocation number is present.
;11520      11608  3                      !
;11521      11609  3                      IF .PATHDESC[PTH$B_LOCINVOC] EQL 0
;11522      11610  3                       THEN
;11523      11611  4                           BEGIN
;11524      11612  4                           INCR I FROM 0 TO .PRIDTBL[-1] - 1 DO
;11525      11613  5                               BEGIN
;11526      11614  5                               PRID = .PRIDTBL[.I] + TABLEBASE;
;11527      11615  5                               IF .PRID[PRID$B_KIND] EQL PRID$K_CONSTANT
;11528      11616  5                               THEN
;11529      11617  6                                   BEGIN
;11530      11618  6
;11531      11619  6                                   LOCAL
;11532      11620  6                                       TEMP_NAME : REF VECTOR [,BYTE];
;11533      11621  6
;11534      11622  6                                   PATHVECTOR1 = PATHDESC[PTH$A_PATHVECTOR];
;11535      11623  6                                   TEMP_NAME = .PATHVECTOR1[0];
;11536      11624  6                                   IF CH$EQL(.PRID[PRID$B_LENGTH],
;11537      11625  6                                             PRID[PRID$A_NAME],
;11538      11626  6                                             .TEMP_NAME[0],
;11539      11627  6                                             TEMP_NAME[1])
;11540      11628  6                                   THEN
;11541      11629  7                                       BEGIN
;11542      11630  7                                       PRIMPTR = CREATE_PRID_CONSTANT(.PRID);
;11543      11631  7                                       RETURN .PRIMPTR;
;11544      11632  6                                       END;
;11545      11633  6
;11546      11634  5                                   END;
;11547      11635  5
;11548      11636  4                               END;
;11549      11637  4
;11550      11638  3                       END;
;11551      11639  3
;11552      11640  3          IF .DBG$GB_LANGUAGE EQL DBG$K_COBOL
;11553      11641  3          THEN
;11554      11642  3              DBG$NCOB_PATHDESC_TO_CS(.PATHDESC, PATHSTRING)
;11555      11643  3          ELSE
;11556      11644  3              DBG$NPATHDESC_TO_CS(.PATHDESC, PATHSTRING);
```

```
;11557          11645  3                SIGNAL(DBG$_NOSYMBOL, 1, .PATHSTRING);
;11558          11646  2                END;
;11559          11647  2
;11560          11648  2
;11561          11649  2              ! Handle the case where the symbol is not unique.  Signal the appro-
;11562          11650  2              ! priate error.
;11563          11651  2              !
;11564          11652  2              [RST$K_NOTUNIQUE,
;11565          11653  2               RST$K_OVERLOAD]:
;11566          11654  3                  BEGIN
;11567          11655  3
;11568          11656  3                  IF .DBG$GB_LANGUAGE EQL DBG$K_COBOL
;11569          11657  3                  THEN
;11570          11658  3                      DBG$NCOB_PATHDESC_TO_CS(.PATHDESC, PATHSTRING)
;11571          11659  3                  ELSE
;11572          11660  3                      DBG$NPATHDESC_TO_CS(.PATHDESC, PATHSTRING);
;11573          11661  3
;11574          11662  3                  IF .KIND EQL RST$K_NOTUNIQUE
;11575          11663  3                  THEN
;11576          11664  3                      SIGNAL(DBG$_NOUNIQUE, 1, .PATHSTRING)
;11577          11665  3                  ELSE
;11578          11666  3                      SIGNAL(DBG$_NOTUNQOVR, 1, .PATHSTRING);
;11579          11667  3
;11580          11668  2                  END;
;11581          11669  2
;11582          11670  2
;11583          11671  2              ! Handle all lexical entities, instruction labels, and line numbers.
;11584          11672  2              ! Zero the FCODE and TYPEID--they do not apply in these cases.
;11585          11673  2              !
;11586          11674  2              [RST$K_ROUTINE,
;11587          11675  2               RST$K_BLOCK,
;11588          11676  2               RST$K_ENTRY,
;11589          11677  2               RST$K_LABEL,
;11590          11678  2               RST$K_LINE]:
;11591          11679  3                  BEGIN
;11592          11680  3
;11593          11681  3                  PRIMPTR[DBG$B_DHDR_KIND] = .KIND;
;11594          11682  3                  PRIMPTR[DBG$L_DHDR_SYMIDO] = .SYMID;
;11595          11683  3                  FCODE = 0;
;11596          11684  3                  TYPEID = 0;
;11597          11685  3
;11598          11686  3
;11599          11687  3                  ! Build a Primary Descriptor Sub-Node for the symbol we have so far.
;11600          11688  3                  !
;11601          11689  3                  DBG$BUILD_PRIMARY_SUBNODE(.PRIMPTR, .KIND, .SYMID, .FCODE, .TYPEID, 0);
;11602          11690  2                  END;
;11603          11691  2
;11604          11692  2
;11605          11693  2              ! Handle data items.  Here we get the FCODE and TYPEID of the data
;11606          11694  2              ! item and store them in the Primary Descriptor.
;11607          11695  2              !
;11608          11696  2              [RST$K_DATA]:
;11609          11697  3                  BEGIN
;11610          11698  3
;11611          11699  3
;11612          11700  3                  ! Walk the up-scope chain collecting all data symids above this one.
;11613          11701  3                  ! This is in order to handle a case such as "X.Y.Z". In this example,
```

```
;11614          11702  3          ! we build a vector of 3 symids: one for Z, one for Y, and one for X.
;11615          11703  3          !
;11616          11704  3          INDEX = -1;
;11617          11705  3          WHILE .KIND EQL RST$K_DATA DO
;11618          11706  4              BEGIN
;11619          11707  4              INDEX = .INDEX + 1;
;11620          11708  4              IF .INDEX GEQ DBG$K_PATHNAME_SIZE
;11621          11709  4              THEN
;11622          11710  4                  $DBG_ERROR('DBGPARSER\PATHNAME_TO_PRIMARY symid stack overflow');
;11623          11711  4              SYMID_VECT[.INDEX] = .SYMID;
;11624          11712  4              SYMID = .SYMID[RST$L_UPSCOPEPTR];
;11625          11713  4              KIND = .SYMID[RST$B_KIND];
;11626          11714  3              END;
;11627          11715  3
;11628          11716  3
;11629          11717  3          ! Now walk back down the symid list, building up the Primary Descriptor.
;11630          11718  3          !
;11631          11719  3          SUBSCR_INDEX = 0;
;11632          11720  3          EXPECTED_SUBS = 0;
;11633          11721  3          TOOFEWSUB = FALSE;
;11634          11722  3          DECR I FROM .INDEX TO 0 DO
;11635          11723  4              BEGIN
;11636          11724  4              SYMID = .SYMID_VECT[.I];
;11637          11725  4
;11638          11726  4              ! Fill in the root node SYMID if we are at the top of the list.
;11639          11727  4              !
;11640          11728  5              IF (.I EQL .INDEX) AND (.PLIPTR EQL 0)
;11641          11729  4              THEN
;11642          11730  4                  PRIMPTR[DBG$L_DHDR_SYMID0] = .SYMID;
;11643          11731  4
;11644          11732  4              ! Get the KIND, FCODE, and TYPEID, and build a new subnode.
;11645          11733  4              !
;11646          11734  4              KIND = .SYMID[RST$B_KIND];
;11647          11735  4              DBG$STA_SETCONTEXT(.SYMID);
;11648          11736  4              DBG$STA_SYMTYPE(.SYMID, FCODE, TYPEID);
;11649          11737  4              DBG$BUILD_PRIMARY_SUBNODE(.PRIMPTR, .KIND, .SYMID, .FCODE, .TYPEID, 0);
;11650          11738  4
;11651          11739  4              ! Obtain a pointer to the newly-built subnode.
;11652          11740  4              !
;11653          11741  4              NODEPTR = .PRIMPTR[DBG$L_PRIM_BLINK];
;11654          11742  4
;11655          11743  4              ! If there was an address override as in P->X then set a flag
;11656          11744  4              ! saying not to use the SYMID of X for address computations
;11657          11745  4              ! (we still retain it for printing, however)
;11658          11746  4              !
;11659          11747  5              IF (.I EQL .INDEX) AND (.PLIPTR NEQ 0)
;11660          11748  4              THEN
;11661          11749  4                  IF .FCODE EQL RST$K_TYPE_RECORD
;11662          11750  4                  THEN
;11663          11751  4                      NODEPTR[DBG$L_PNODE_SYMID] = 0
;11664          11752  4                  ELSE
;11665          11753  4                      NODEPTR[DBG$V_PNODE_IGNORE] = TRUE;
;11666          11754  4
;11667          11755  4              ! If we have just attached an array subnode then fill
;11668          11756  4              ! in the subscript inf rmation here.
;11669          11757  4              !
;11670          11758  4              IF .FCODE EQL RST$K_TYPE_ARRAY
```

```
;11671          11759  4                AND .COMPONENTS_IN_PATHNAME
;11672          11760  4                THEN
;11673          11761  5                    BEGIN
;11674          11762  5                    SUBVECTOR = NODEPTR[DBG$A_PNARR_SVECTOR];
;11675          11763  5                    EXPECTED_SUBS = .EXPECTED_SUBS + .NODEPTR[DBG$B_PNARR_DIMCNT];
;11676          11764  5
;11677          11765  5                    ! Loop through the dimensions of the array.
;11678          11766  5                    !
;11679          11767  5                    INCR J FROM 0 TO .NODEPTR[DBG$B_PNARR_DIMCNT]-1 DO
;11680          11768  6                        BEGIN
;11681          11769  6
;11682          11770  6                        ! Signal an error if not enough subscripts were supplied.
;11683          11771  6                        !
;11684          11772  7                        IF  (.SUBSCR_INDEX GEQ .SUBSCR_DESC[0, SUBSCR$B_SUBCNT])
;11685          11773  7                        AND ((.SUBSCR_DESC[0, SUBSCR$B_SUBCNT] NEQ 0) OR
;11686          11774  7                            (.I NEQ 0))
;11687          11775  6                        THEN
;11688          11776  7                            BEGIN
;11689          11777  7
;11690          11778  7                            ! We have a problem in that we know we don't have
;11691          11779  7                            ! enough subscripts, but we don't know exactly
;11692          11780  7                            ! how many we were expecting. So what we do here
;11693          11781  7                            ! is just set a flag saying to signal the error
;11694          11782  7                            ! later on (after we do know).
;11695          11783  7                            !
;11696          11784  7                            TOOFEWSUB = TRUE;
;11697          11785  7                            SUBSCR_DESC[0, SUBSCR$B_SUBCNT] = 0;
;11698          11786  6                            END;
;11699          11787  6
;11700          11788  6                        ! Special check for no subscripts specified - treat this
;11701          11789  6                        ! the same as if (*,*,...) were specified (aggregate
;11702          11790  6                        ! examine).
;11703          11791  6                        !
;11704          11792  6                        IF .SUBSCR_DESC[0, SUBSCR$B_SUBCNT] EQL 0
;11705          11793  6                        THEN
;11706          11794  6                            INCR J FROM 0 TO .NODEPTR[DBG$B_PNARR_DIMCNT] DO
;11707          11795  7                                BEGIN
;11708          11796  7                                SUBSCR_DESC[.SUBSCR_INDEX + .J,
;11709          11797  7                                    SUBSCR$V_RANGE] = TRUE;
;11710          11798  7                                SUBSCR_DESC[.SUBSCR_INDEX + .J,
;11711          11799  7                                    SUBSCR$V_ASTER] = TRUE;
;11712          11800  6                                END;
;11713          11801  6
;11714          11802  6
;11715          11803  6                        ! If we were given a range then fix up the sub-node
;11716          11804  6                        ! to reflect a ranged examine.
;11717          11805  6                        !
;11718          11806  6                        IF .SUBSCR_DESC[.SUBSCR_INDEX, SUBSCR$V_RANGE]
;11719          11807  6                        THEN
;11720          11808  7                            BEGIN
;11721          11809  7
;11722          11810  7                            ! Do not allow asterisk unless we are at the
;11723          11811  7                            ! bottom level. *** TEMPORARY
;11724          11812  7                            !
;11725          11813  7                            IF (.I NEQ 0) AND
;11726          11814  8                               (.SUBSCR_DESC[0, SUBSCR$B_SUBCNT] NEQ 0)
;11727          11815  7                            THEN
```

```
;11728      11816  7      IF .SUBSCR_DESC[.SUBSCR_INDEX, SUBSCR$V_ASTER]
;11729      11817  7      THEN
;11730      11818  7          SIGNAL(DBG$_ILLASTER)
;11731      11819  7      ELSE
;11732      11820  7          SIGNAL(DBG$_ILLRANGE);
;11733      11821  7
;11734      11822  7
;11735      11823  7  IF NOT .NODEPTR[DBG$V_PNARR_RANGE]
;11736      11824  7  THEN
;11737      11825  8      BEGIN
;11738      11826  8      NODEPTR[DBG$V_PNARR_RANGE] = TRUE;
;11739      11827  8      INCR K FROM 0 TO .J-1 DO
;11740      11828  9          BEGIN
;11741      11829  9          SUBVECTOR[.K, DBG$L_PNSUB_LBOUND] =
;11742      11830  9              .SUBVECTOR[.K, DBG$L_PNSUB_SVALUE];
;11743      11831  9          SUBVECTOR[.K, DBG$L_PNSUB_UBOUND] =
;11744      11832  9              .SUBVECTOR[.K, DBG$L_PNSUB_SVALUE];
;11745      11833  8          END;
;11746      11834  7      END;
;11747      11835  7
;11748      11836  7  IF NOT .SUBSCR_DESC[.SUBSCR_INDEX, SUBSCR$V_ASTER]
;11749      11837  7  THEN
;11750      11838  8      BEGIN
;11751      11839  8
;11752      11840  8      ! Check for reversed range.
;11753      11841  8      !
;11754      11842  8      IF .SUBSCR_DESC[.SUBSCR_INDEX, SUBSCR$L_LBOUND] GTR
;11755      11843  8         .SUBSCR_DESC[.SUBSCR_INDEX, SUBSCR$L_UBOUND]
;11756      11844  8      THEN
;11757      11845  8          SIGNAL(DBG$_INVRANSPEC);
;11758      11846  8
;11759      11847  8      SUBVECTOR[.J, DBG$L_PNSUB_LBOUND] =
;11760      11848  8          .SUBSCR_DESC[.SUBSCR_INDEX, SUBSCR$L_LBOUND];
;11761      11849  8      SUBVECTOR[.J, DBG$L_PNSUB_UBOUND] =
;11762      11850  8          .SUBSCR_DESC[.SUBSCR_INDEX, SUBSCR$L_UBOUND];
;11763      11851  7      END;
;11764      11852  7
;11765      11853  7  SUBVECTOR[.J, DBG$L_PNSUB_SVALUE] =
;11766      11854  7      .SUBVECTOR[.J, DBG$L_PNSUB_LBOUND]
;11767      11855  7  END
;11768      11856  7
;11769      11857  7  ! Fill in the subscript value.
;11770      11858  7  !
;11771      11859  6  ELSE
;11772      11860  7      BEGIN
;11773      11861  7      LOCAL
;11774      11862  7          VAL;
;11775      11863  7
;11776      11864  7      VAL = .SUBSCR_DESC[.SUBSCR_INDEX, SUBSCR$L_LBOUND];
;11777      11865  7      IF (.VAL LSS .SUBVECTOR[.J, DBG$L_PNSUB_LBOUND]) OR
;11778      11866  8         (.VAL GTR .SUBVECTOR[.J, DBG$L_PNSUB_UBOUND])
;11779      11867  7      THEN
;11780      11868  7          SIGNAL(DBG$_SUBOUTBND, 4, .SUBSCR_INDEX, .VAL,
;11781      11869  7                 .SUBVECTOR[.J, DBG$L_PNSUB_LBOUND],
;11782      11870  7                 .SUBVECTOR[.J, DBG$L_PNSUB_UBOUND]);
;11783      11871  7      SUBVECTOR[.J, DBG$L_PNSUB_SVALUE] =
;11784      11872  7          .SUBSCR_DESC[.SUBSCR_INDEX, SUBSCR$L_LBOUND];
```

```
;11785          11873  6                    END;
;11786          11874  6
;11787          11875  6                ! If we previously got a range, then make this dimension
;11788          11876  6                ! into a range also.
;11789          11877  6                !
;11790          11878  6                IF .NODEPTR[DBG$V_PNARR_RANGE] AND
;11791          11879  6                NOT .SUBSCR_DESC[.SUBSCR_INDEX, SUBSCR$V_RANGE]
;11792          11880  6                THEN
;11793          11881  7                    BEGIN
;11794          11882  7                    SUBVECTOR[.J, DBG$L_PNSUB_LBOUND] =
;11795          11883  7                        .SUBVECTOR[.J, DBG$L_PNSUB_SVALUE];
;11796          11884  7                    SUBVECTOR[.J, DBG$L_PNSUB_UBOUND] =
;11797          11885  7                        .SUBVECTOR[.J, DBG$L_PNSUB_SVALUE];
;11798          11886  6                    END;
;11799          11887  6
;11800          11888  6                SUBSCR_INDEX = .SUBSCR_INDEX + 1;
;11801          11889  5                END;
;11802          11890  5
;11803          11891  5            ! Fill in the field that gives the count of subscripts.
;11804          11892  5            ! Also, unless this is an array slice, then light the
;11805          11893  5            ! EVAL bit which says that subscripting is being done,
;11806          11894  5            ! and tack on a new subnode.
;11807          11895  5            !
;11808          11896  5            NODEPTR[DBG$B_PNARR_SUBCNT] = .NODEPTR[DBG$B_PNARR_DIMCNT];
;11809          11897  5            IF NOT .NODEPTR[DBG$V_PNARR_RANGE]
;11810          11898  5            THEN
;11811          11899  6                BEGIN
;11812          11900  6                NODEPTR[DBG$V_PNODE_EVAL] = TRUE;
;11813          11901  6
;11814          11902  6                ! Attach a new Primary sub-node for the array elements.
;11815          11903  6                !
;11816          11904  6                TYPEID = .NODEPTR[DBG$L_PNARR_CELLTYPE];
;11817          11905  6                FCODE = DBG$STA_TYPEFCODE(.TYPEID);
;11818          11906  6                DBG$BUILD_PRIMARY_SUBNODE (.PRIMPTR, RST$K_DATA, 0,
;11819          11907  6                    .FCODE, .TYPEID, 0);
;11820          11908  6                NODEPTR = .PRIMPTR[DBG$L_PRIM_BLINK];
;11821          11909  5                END;
;11822          11910  4            END;
;11823          11911  4
;11824          11912  4
;11825          11913  4            ! If we have just attached a record subnode then fill
;11826          11914  4            ! in the component information here. Note that this must
;11827          11915  4            ! be done after the array case above, to properly handle
;11828          11916  4            ! arrays of records.
;11829          11917  4            !
;11830          11918  4            IF (.FCODE EQL RST$K_TYPE_RECORD) AND
;11831          11919  4               (.COMPONENTS_IN_PATHNAME) AND
;11832          11920  5               (.I NEQ 0)
;11833          11921  4            THEN
;11834          11922  5                BEGIN
;11835          11923  5
;11836          11924  5                ! If we are not at the bottom symid, then there must be
;11837          11925  5                ! a symid for the record component below this one.
;11838          11926  5                ! We need to fill in the corresponding record sub-node
;11839          11927  5                ! with the component index. This component index is
;11840          11928  5                ! used by MODIFY_PRIMARY to compute logical sucessor/
;11841          11929  5                ! predecessor. We also light the "EVAL" bit in the
```

```
;11842      11930  5                    ! sub-node.
;11843      11931  5                    !
;11844      11932  5                    NODEPTR[DBG$V_PNODE_EVAL] = TRUE;
;11845      11933  5                    SYMID1 = .SYMID_VECT[.I-1];
;11846      11934  5                    TYPCOMPLST = TYPEID[RST$A_TYPCOMPLST];
;11847      11935  5                    INCR J FROM 0 TO .TYPEID[RST$L_TYPCOMPCNT] - 1 DO
;11848      11936  6                        BEGIN
;11849      11937  6                        SYMID2 = .TYPCOMPLST[.J];
;11850      11938  6
;11851      11939  6
;11852      11940  6                        ! Use the DSTPTR to determine whether we are
;11853      11941  6                        ! really looking at the right component.
;11854      11942  6                        !
;11855      11943  6                        IF .SYMID1[RST$L_DSTPTR] EQL .SYMID2[RST$L_DSTPTR]
;11856      11944  6                        THEN
;11857      11945  7                            BEGIN
;11858      11946  7                            NODEPTR[DBG$W_PNREC_INDEX] = .J + 1;
;11859      11947  7                            EXITLOOP;
;11860      11948  6                            END;
;11861      11949  6
;11862      11950  6                        ! We should not fall through to here.
;11863      11951  5                        END;
;11864      11952  4                    END;
;11865      11953  3                END;
;11866      11954  3
;11867      11955  3            ! If we have not exhausted the given list of subscripts,
;11868      11956  3            ! then first check for a substring reference.
;11869      11957  3            ! We must have seen something that
;11870      11958  3            ! looks like a ranged subscript "(i:j)". In order for a
;11871      11959  3            ! substring to be legal then the data type must
;11872      11960  3            ! either be text or one of the decimal string types.
;11873      11961  3            !
;11874      11962  3            PICKED_UP_SUBSTRING = 0;
;11875      11963  4            IF  (.SUBSCR_INDEX EQL (.SUBSCR_DESC[0, SUBSCR$B_SUBCNT]-1))
;11876      11964  3            AND .SUBSCR_DESC[.SUBSCR_INDEX, SUBSCR$V_RANGE]
;11877      11965  3            AND .SUBSCR_DESC[.SUBSCR_INDEX, SUBSCR$V_MARKER]
;11878      11966  4            AND (NOT .SUBSCR_DESC[.SUBSCR_INDEX, SUBSCR$V_ASTER])
;11879      11967  3            THEN
;11880      11968  4                BEGIN
;11881      11969  4                DTYPE = DSC$K_DTYPE_Z;
;11882      11970  4                IF .KIND EQL RST$K_DATA
;11883      11971  4                THEN
;11884      11972  5                    BEGIN
;11885      11973  5                    SELECTONE .FCODE OF
;11886      11974  5                        SET
;11887      11975  5                        [RST$K_TYPE_ATOMIC]:
;11888      11976  6                            BEGIN
;11889      11977  6                            DBG$STA_TYP_ATOMIC(.TYPEID, DTYPE, BITSIZE);
;11890      11978  6                            LEN = .BITSIZE / 8;
;11891      11979  5                            END;
;11892      11980  5                        [RST$K_TYPE_DESCR]:
;11893      11981  6                            BEGIN
;11894      11982  6                            DBG$STA_TYP_DESCR(.TYPEID, DESCR);
;11895      11983  6                            DTYPE = .DESCR[DSC$B_DTYPE];
;11896      11984  6                            LEN = .DESCR[DSC$W_LENGTH];
;11897      11985  5                            END;
;11898      11986  5                        [RST$K_TYPE_PICT, RST$K_TYPE_RECORD]:
```

```
;11899          11987  6                    BEGIN
;11900          11988  6                    DTYPE = DSC$K_DTYPE_T;
;11901          11989  6                    LEN = 10000;            ! Pick large number to allow
;11902          11990  6                                            ! arbitrarily long substring.
;11903          11991  5                    END;
;11904          11992  5                TES;
;11905          11993  5            IF (.DTYPE EQL DSC$K_DTYPE_T) OR
;11906          11994  5               (.DTYPE EQL DSC$K_DTYPE_NU) OR
;11907          11995  5               (.DTYPE EQL DSC$K_DTYPE_NL) OR
;11908          11996  5               (.DTYPE EQL DSC$K_DTYPE_NLO) OR
;11909          11997  5               (.DTYPE EQL DSC$K_DTYPE_NR) OR
;11910          11998  5               (.DTYPE EQL DSC$K_DTYPE_NRO) OR
;11911          11999  6               (.DTYPE EQL DSC$K_DTYPE_NZ)
;11912          12000  5            THEN
;11913          12001  6                BEGIN
;11914          12002  6
;11915          12003  6                ! Modify the primary to indicate the substring information
;11916          12004  6                !
;11917          12005  6                PRIMPTR [DBG$V_DHDR_AGGR] = FALSE;
;11918          12006  6                PRIMPTR [DBG$V_DHDR_SUBREF] = TRUE;
;11919          12007  6
;11920          12008  6                IF .SUBSCR_DESC[.SUBSCR_INDEX, SUBSCR$L_LBOUND] GTR
;11921          12009  6                   %X'7FFFF'
;11922          12010  6                THEN
;11923          12011  6                    SIGNAL(DBG$_ILLOFFSET, 1,
;11924          12012  6                           .SUBSCR_DESC[.SUBSCR_INDEX, SUBSCR$L_LBOUND]);
;11925          12013  6                PRIMPTR [DBG$W_PRIM_OFFSET] =
;11926          12014  6                    .SUBSCR_DESC[.SUBSCR_INDEX, SUBSCR$L_LBOUND];
;11927          12015  8                IF ((.SUBSCR_DESC[.SUBSCR_INDEX, SUBSCR$L_LBOUND] +
;11928          12016  8                     .SUBSCR_DESC[.SUBSCR_INDEX, SUBSCR$L_UBOUND]-1)
;11929          12017  6                     GTR .LEN) OR
;11930          12018  7                    (.SUBSCR_DESC[.SUBSCR_INDEX, SUBSCR$L_LBOUND]
;11931          12019  7                     LSS 1)
;11932          12020  6                THEN
;11933          12021  6                    SIGNAL(DBG$_SUBSTRING, 3,
;11934          12022  6                           .SUBSCR_DESC[.SUBSCR_INDEX, SUBSCR$L_LBOUND],
;11935          12023  6                           .SUBSCR_DESC[.SUBSCR_INDEX, SUBSCR$L_UBOUND],
;11936          12024  6                           .LEN);
;11937          12025  6
;11938          12026  6                IF .SUBSCR_DESC[.SUBSCR_INDEX, SUBSCR$L_UBOUND] GTR
;11939          12027  6                       %X'7FFF'
;11940          12028  6                THEN
;11941          12029  6                    SIGNAL(DBG$_ILLSUBLEN, 1,
;11942          12030  6                           .SUBSCR_DESC[.SUBSCR_INDEX, SUBSCR$L_UBOUND]);
;11943          12031  6                PRIMPTR [DBG$W_PRIM_LENGTH] =
;11944          12032  6                    .SUBSCR_DESC[.SUBSCR_INDEX, SUBSCR$L_UBOUND];
;11945          12033  6                NODEPTR = .PRIMPTR [DBG$L_PRIM_BLINK];
;11946          12034  6                NODEPTR [DBG$L_PNODE_RELOC] = -1;
;11947          12035  6                PICKED_UP_SUBSTRING = 1;
;11948          12036  6                SUBSCR_INDEX = .SUBSCR_INDEX + 1;
;11949          12037  6                END
;11950          12038  5            ELSE
;11951          12039  5
;11952          12040  5                ! We have seen something that looks like a substring
;11953          12041  5                ! but the data type is wrong. Catch that case here and
;11954          12042  5                ! signal an error.
;11955          12043  5                !
```

```
;11956          12044  5                    SIGNAL(DBG$_ILLSUBSTR);
;11957          12045  5                END
;11958          12046  5
;11959          12047  4            ELSE
;11960          12048  4                ! We have seen something that looks like a substring
;11961          12049  4                ! but the data type is wrong. Catch that case here and
;11962          12050  4                ! signal an error.
;11963          12051  4                !
;11964          12052  4                SIGNAL(DBG$_ILLSUBSTR);
;11965          12053  3            END;
;11966          12054  3
;11967          12055  3
;11968          12056  3        ! Signal an error if we picked up too many or too few subscripts.
;11969          12057  3        !
;11970          12058  3        IF .TOOFEWSUB
;11971          12059  3        THEN
;11972          12060  3            SIGNAL(DBG$_TOOFEWSUB, 1, .EXPECTED_SUBS);
;11973          12061  3        IF .SUBSCR_INDEX LSS .SUBSCR_DESC[0, SUBSCR$B_SUBCNT]
;11974          12062  3        THEN
;11975          12063  3            SIGNAL(DBG$_TOOMANSUB, 1, .SUBSCR_INDEX-.PICKED_UP_SUBSTRING);
;11976          12064  3
;11977          12065  2        END;
;11978          12066  2
;11979          12067  2    ! Anything else we treat as an internal DEBUG error.
;11980          12068  2    !
;11981          12069  2    [OTHERWISE]:
;11982          12070  2        $DBG_ERROR('DBGPARSER\PATHNAME_TO_PRIMARY 10');
;11983          12071  2
;11984          12072  2    TES;
;11985          12073  2
;11986          12074  2
;11987          12075  2    ! Return a pointer to the Primary Descriptor to the caller.
;11988          12076  2    !
;11989          12077  2    RETURN .PRIMPTR;
;11990          12078  1    END;


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

48  54  41  50  5C  52  45  53  52  41  50  47  42  44  32  033F9 P.AYR:  .ASCII  \2DBGPARSER\<92>\PATHNAME_TO_PRIMARY sym\  ;
59  52  41  4D  49  52  50  5F  4F  54  5F  45  4D  41  4E  03408                                                            ;
                                        6D  79  73  20  03417                                                            ;
6C  66  72  65  76  6F  20  6B  63  61  74  73  20  64  69  0341B          .ASCII  \id stack overflow\                       ;
                                                    77  6F  0342A                                                        ;
48  54  41  50  5C  52  45  53  52  41  50  47  42  44  20  0342C P.AYS:  .ASCII  \ DBGPARSER\<92>\PATHNAME_TO_PRIMARY 10\  ;
59  52  41  4D  49  52  50  5F  4F  54  5F  45  4D  41  4E  0343B                                                            ;
                                            30  31  20  0344A                                                            ;


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

                                                OFFC 00000 PATHNAME_TO_PRIMARY:
                                                           .WORD   Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11        ; 1141
                                    5E      FEE4    CE  9E 00002           MOVAB   -284(SP), SP                        ;
                                    58        04    AC  D0 00007           MOVL    PATHDESC, R8                        ; 1152
```

```
                                59        10    AC   D0 0000B          MOVL    SAVED_PATHDESC, R9                          ; 1152
                                          04    AE   D4 0000F          CLRL    4(SP)                                       ; 1151
                                          0C    AC   D5 00012          TSTL    PLIPTR                                      ;
                                                50   13 00015          BEQL    5$                                          ;
                                          04    AE   D6 00017          INCL    4(SP)                                       ;
                                5B        0C    AC   D0 0001A          MOVL    PLIPTR, PRIMPTR                             ; 1151
                                56        18    AB   D0 0001E          MOVL    24(PRIMPTR), NODEPTR                        ; 1151
                          0A    A6              01   88 00022          BISB2   #1, 10(NODEPTR)                             ; 1151
                                01        01    A8   91 00026          CMPB    1(R8), #1                                   ; 1152
                                                67   1A 0002A          BGTRU   6$                                          ;
                                01        01    A9   91 0002C          CMPB    1(R9), #1                                   ; 1152
                                                6A   1B 00030          BLEQU   8$                                          ;
                                57        08    A8   9E 00032          MOVAB   8(R8), PATHVECTOR1                          ; 1153
                                52        08    A9   9E 00036          MOVAB   8(R9), PATHVECTOR2                          ; 1153
                                51        01    A9   9A 0003A          MOVZBL  1(R9), I                                    ; 1153
                                                51   D7 0003E          DECL    I                                           ;
                                50                    68   9A 00040          MOVZBL  (R8), J                             ; 1154
                                                      09   11 00043          BRB     2$                                  ;
                          53            51            50   C1 00045 1$:      ADDL3   J, I, R3                            ;
                                      6743          6740   D0 00049          MOVL    (PATHVECTOR1)[J], (PATHVECTOR1)[R3] ;
                                        F4            50   F4 0004E 2$:      SOBGEQ  J, 1$                               ;
                                        50            51   D0 00051          MOVL    I, J                                ; 1154
                                                      05   11 00054          BRB     4$                                  ;
                                      6740          6240   D0 00056 3$:      MOVL    (PATHVECTOR2)[J], (PATHVECTOR1)[J]  ;
                                        F8            50   F4 0005B 4$:      SOBGEQ  J, 3$                               ;
                                        68            51   80 0005E          ADDB2   I, (R8)                             ; 1154
                                01      A8            51   80 00061          ADDB2   I, 1(R8)                            ; 1154
                                                      35   11 00065          BRB     8$                                  ; 1151
                                                      09   DD 00067 5$:      PUSHL   #9                                  ; 1155
                    00000000G   00                    01   FB 00069          CALLS   #1, DBG$GET_TEMPMEM                 ;
                                5B                    50   D0 00070          MOVL    R0, PRIMPTR                         ;
                                      02    AB              79   8F 00073          MOVB    #121, 2(PRIMPTR)            ; 1155
                                      03    AB  00000000G   00   90 00078          MOVB    DBG$GB_LANGUAGE, 3(PRIMPTR) ; 1155
                                      6B                    24   B0 00080          MOVW    #36, (PRIMPTR)              ; 1155
                                      14    AB        14    AB   9E 00083          MOVAB   20(PRIMPTR), 20(PRIMPTR)    ; 1156
                                      18    AB        14    AB   9E 00088          MOVAB   20(PRIMPTR), 24(PRIMPTR)    ; 1156
                                      01              01    A8   91 0008D          CMPB    1(R8), #1                   ; 1156
                                                            06   1B 00091          BLEQU   7$                          ;
                              69                    68          34   28 00093 6$:      MOVC3   #52, (R8), (R9)         ; 1156
                                                            03   11 00097          BRB     8$                          ;
                                                      01    A9   94 00099 7$:      CLRB    1(R9)                       ; 1156
                                5B        0009C       00    5B   D0 0009C 8$:      MOVL    PRIMPTR, DBG$GL_CURRENT_PRIMARY ; 1157
                                          0C  00000000' EF    E9 000A3          BLBC    COMPONENTS_IN_PATHNAME, 9$  ; 1158
                                                      50   D4 000AA          CLRL    R0                          ; 1158
                00      08    BC                    08          08   ED 000AC          CMPZV   #8, #8, @SUBSCR_DESC, #0    ;
                                                      09   14 000B2          BGTR    10$                         ;
                                                      09   11 000B4          BRB     11$                         ;
                                                      50   D4 000B6 9$:      CLRL    R0                          ; 1158
                                                04          6C   91 000B8          CMPB    (AP), #4                    ;
                                                            02   1B 000BB          BLEQU   11$                         ;
                                                      50   D6 000BD 10$:      INCL    R0                          ;
                                                51          50   D0 000BF 11$:      MOVL    R0, ARR_FLAG                ;
                                                            7E   D4 000C2          CLRL    -(SP)                       ; 1158
                                                            51   DD 000C4          PUSHL   ARR_FLAG                    ;
                                                      2C    AE   9F 000C6          PUSHAB  SCOPE                       ;
                                                      34    AE   9F 000C9          PUSHAB  SCOPE_STATE                 ;
                                                      3C    AE   9F 000CC          PUSHAB  KIND                        ;
```

```
                                 44   AE  9F 000CF          PUSHAB  SYMID                                   ;
                                 58   DD 000D2            PUSHL   R8                                      ;
                  00000000G      00   07  FB 000D4        CALLS   #7, DBG$STA_GETSYMBOL                   ;
                         1C      AB   28  AE  90 000DB     MOVB    SCOPE_STATE, 28(PRIMPTR)                ; 1158
                         20      AB   24  AE  D0 000E0     MOVL    SCOPE, 32(PRIMPTR)                      ; 1158
                                 53   2C  AE  D0 000E5     MOVL    KIND, R3                                ; 1159
                                      03  13 000E9        BEQL    12$                                     ; 1159
                                    0083  31 000EB        BRW     18$                                     ;
                                      02  A8  95 000EE 12$: TSTB    2(R8)                                   ; 1160
                                      4E  12 000F1        BNEQ    15$                                     ;
                                 50 00000000'  EF  D0 000F3 MOVL    PRIDTBL, R0                             ; 1161
                                 59   FC  A0  D0 000FA     MOVL    -4(R0), R9                              ;
                                 55       01  CE 000FE     MNEGL   #1, I                                   ;
                                          3A  11 00101     BRB     14$                                     ;
                                 50 00000000'  EF  9E 00103 13$: MOVAB   TABLEBASE, R0                      ; 1161
                  54             50 00000000'FF45  C1 0010A ADDL3   @PRIDTBL[I], R0, PRID                  ;
                                 01       64  91 00113     CMPB    (PRID), #1                              ; 1161
                                          25  12 00116     BNEQ    14$                                     ;
                                 57   08  A8  9E 0011      MOVAB   8(R8), PATHVECTOR1                      ; 1162
                                 50       67  D0 0011C     MOVL    (PATHVECTOR1), TEMP_NAME                ; 1162
                                 52   08  A4  9A 0011F     MOVZBL  8(PRID), R2                             ; 1162
                                 51       60  9A 00123     MOVZBL  (TEMP_NAME), R1                         ; 1162
          51     00       09   A4          52  2D 00126     CMPC5   R2, 9(PRID), #0, R1, 1(TEMP_NAME)       ; 1162
                                      01  A0       0012C                                                   ;
                                          0D  12 0012E     BNEQ    14$                                     ;
                                          54  DD 00130     PUSHL   PRID                                    ; 1163
                                EA65      CF      01  FB 00132 CALLS   #1, CREATE_PRID_CONSTANT           ;
                                 5B           50  D0 00137 MOVL    R0, PRIMPTR                             ;
                                            050D  31 0013A BRW     84$                                     ; 1163
                  C2             55       59  F2 0013D 14$: AOBLSS  R9, I, 13$                             ; 1161
                                 03 00000000G  00  91 00141 15$: CMPB    DBG$GB_LANGUAGE, #3                 ; 1164
                                          0E  12 00148     BNEQ    16$                                     ;
                                 34   AE  9F 0014A         PUSHAB  PATHSTRING                              ; 1164
                                 58   DD 0014D            PUSHL   R8                                      ;
  00000000G      00                    02  FB 0014F        CALLS   #2, DBG$NCOB_PATHDESC_TO_CS             ;
                                          0C  11 00156     BRB     17$                                     ;
                                 34   AE  9F 00158 16$:    PUSHAB  PATHSTRING                              ; 1164
                                 58   DD 0015B            PUSHL   R8                                      ;
  00000000G      00                    02  FB 0015D        CALLS   #2, DBG$NPATHDESC_TO_CS                 ;
                                 34   AE  DD 00164 17$:    PUSHL   PATHSTRING                              ; 1164
                                 01   DD 00167            PUSHL   #1                                      ;
                                    000281F8  8F  DD 00169 PUSHL   #164344                                 ;
                                          4A  11 0016F     BRB     23$                                     ;
                                 09       53  D1 00171 18$: CMPL    R3, #9                                  ; 1165
                                          05  13 00174     BEQL    19$                                     ;
                                 0D       53  D1 00176     CMPL    R3, #13                                 ;
                                          43  12 00179     BNEQ    24$                                     ;
                                 03 00000000G  00  91 0017B 19$: CMPB    DBG$GB_LANGUAGE, #3                 ; 1165
                                          0E  12 00182     BNEQ    20$                                     ;
                                 34   AE  9F 00184         PUSHAB  PATHSTRING                              ; 1165
                                 58   DD 00187            PUSHL   R8                                      ;
  00000000G      00                    02  FB 00189        CALLS   #2, DBG$NCOB_PATHDESC_TO_CS             ;
                                          0C  11 00190     BRB     21$                                     ;
                                 34   AE  9F 00192 20$:    PUSHAB  PATHSTRING                              ; 1166
                                 58   DD 00195            PUSHL   R8                                      ;
  00000000G      00                    02  FB 00197        CALLS   #2, DBG$NPATHDESC_TO_CS                 ;
                 09                     53  D1 0019E 21$:    CMPL    R3, #9                                  ; 1166
```

```
                                 0D  12 001A1          BNEQ    22$                                     ;
                              34 AE  DD 001A3          PUSHL   PATHSTRING                              ; 1166
                                 01  DD 001A6          PUSHL   #1                                      ;
                       000281F0 8F  DD 001A8          PUSHL   #164336                                 ;
                                 0B  11 001AE          BRB     23$                                     ;
                              34 AE  DD 001B0 22$:      PUSHL   PATHSTRING                              ; 1166
                                 01  DD 001B3          PUSHL   #1                                      ;
                       000282A8 8F  DD 001B5          PUSHL   #164520                                 ;
                           0485  31 001BB 23$:        BRW     83$                                     ;
                        02          53  D1 001BE 24$:      CMPL    R3, #2                              ; 1167
                                    05  19 001C1          BLSS    25$                                 ;
                        05          53  D1 001C3          CMPL    R3, #5                              ;
                                    05  15 001C6          BLEQ    26$                                 ;
                        08          53  D1 001C8 25$:      CMPL    R3, #8                              ;
                                    23  12 001CB          BNEQ    27$                                 ;
                     07 AB          53  90 001CD 26$:      MOVB    R3, 7(PRIMPTR)                      ; 1168
                     0C AB       30 AE  D0 001D1          MOVL    SYMID, 12(PRIMPTR)                  ; 1168
                                 38 AE  7C 001D6          CLRQ    TYPEID                              ; 1168
                                 7E  D4 001D9          CLRL    -(SP)                               ; 1168
                                 3C AE  DD 001DB          PUSHL   TYPEID                              ;
                                 44 AE  DD 001DE          PUSHL   FCODE                               ;
                                 3C AE  DD 001E1          PUSHL   SYMID                               ;
                                    53  DD 001E4          PUSHL   R3                                  ;
                                    5B  DD 001E6          PUSHL   PRIMPTR                             ;
                   C4E8 CF          06  FB 001E8          CALLS   #6, DBG$BUILD_PRIMARY_SUBNODE       ;
                           045A  31 001ED          BRW     84$                                     ; 1159
                        06          53  D1 001F0 27$:      CMPL    R3, #6                              ; 1169
                                    03  13 001F3          BEQL    28$                                 ;
                           043D  31 001F5          BRW     82$                                     ;
                        55          01  CE 001F8 28$:      MNEGL   #1, INDEX                           ; 1170
                        52       30 AE  D0 001FB          MOVL    SYMID, R2                           ; 1171
                        06       2C AE  D1 001FF 29$:      CMPL    KIND, #6                            ; 1170
                                    31  12 00203          BNEQ    31$                                 ;
                                    55  D6 00205          INCL    INDEX                               ; 1170
                        34          55  D1 00207          CMPL    INDEX, #52                          ; 1170
                                    15  19 0020A          BLSS    30$                                 ;
                           00000000' EF  9F 0020C          PUSHAB  P.AYR                               ; 1171
                                    01  DD 00212          PUSHL   #1                                  ;
                           00028362 8F  DD 00214          PUSHL   #164706                             ;
                00000000G   00          03  FB 0021A          CALLS   #3, LIB$SIGNAL                      ;
                      4C AE45           52  D0 00221 30$:      MOVL    R2, SYMID_VECT[INDEX]               ; 1171
                      30 AE       10 A2  D0 00226          MOVL    16(R2), SYMID                       ; 1171
                         52       30 AE  D0 0022B          MOVL    SYMID, R2                           ; 1171
                      2C AE       14 A2  9A 0022F          MOVZBL  20(R2), KIND                        ;
                                    C9  11 00234          BRB     29$                                 ; 1170
                                    54  D4 00236 31$:      CLRL    SUBSCR_INDEX                        ; 1171
                                 1C AE  7C 00238          CLRQ    TOOFEWSUB                           ; 1172
                        53       01 A5  9E 0023B          MOVAB   1(R5), I                            ; 1174
                                 0274  31 0023F          BRW     64$                                 ;
                      30 AE       4C AE43  D0 00242 32$:      MOVL    SYMID_VECT[I], SYMID                ; 1172
                                    59  D4 00248          CLRL    R9                                  ; 1172
                        55          53  D1 0024A          CMPL    I, INDEX                            ;
                                    0C  12 0024D          BNEQ    33$                                 ;
                                    59  D6 0024F          INCL    R9                                  ;
                                 0C AC  D5 00251          TSTL    PLIPTR                              ;
                                    05  12 00254          BNEQ    33$                                 ;
                      0C AB       30 AE  D0 00256          MOVL    SYMID, 12(PRIMPTR)                  ; 1173
```

```
                                57      30   AE  D0 0025B 33$:    MOVL    SYMID, R7                                       ; 1173
                             2C AE      14   A7  9A 0025F         MOVZBL  20(R7), KIND
                                                57  DD 00264         PUSHL   R7                                              ; 1173
                00000000G    00                 01  FB 00266         CALLS   #1, DBG$STA_SETCONTEXT
                                                38  AE  9F 0026D     PUSHAB  TYPEID                                          ; 1173
                                                40  AE  9F 00270     PUSHAB  FCODE
                                                57  DD 00273         PUSHL   R7
                00000000G    00                 03  FB 00275         CALLS   #3, DBG$STA_SYMTYPE
                                                7E  D4 0027C         CLRL    -(SP)                                           ; 1173
                                                3C  AE  DD 0027E     PUSHL   TYPEID
                                                44  AE  DD 00281     PUSHL   FCODE
                                                57  DD 00284         PUSHL   R7
                                                3C  AE  DD 00286     PUSHL   KIND
                                                5B  DD 00289         PUSHL   PRIMPTR
                             C445 CF                06  FB 0028B     CALLS   #6, DBG$BUILD_PRIMARY_SUBNODE
                                  56        18   AB  D0 00290         MOVL    24(PRIMPTR), NODEPTR                            ; 1174
                                  13        59  E9 00294             BLBC    R9, 35$                                         ; 1174
                                  0F        04   AE  E9 00297        BLBC    4(SP), 35$
                                  07        3C   AE  D1 0029B        CMPL    FCODE, #7                                       ; 1174
                                                05  12 0029F         BNEQ    34$
                                                10  A6  D4 002A1     CLRL    16(NODEPTR)                                     ; 1175
                                                04  11 002A4         BRB     35$
                                          0A    A6                  20  88 002A6 34$:   BISB2   #32, 10(NODEPTR)          ; 1175
                                                01  3C   AE  D1 002AA 35$:   CMPL    FCODE, #1                               ; 1175
                                                03  13 002AE         BEQL    37$
                                                    01BB  31 002B0 36$:   BRW     61$
                                                F6 00000000'  EF  E9 002B3 37$:   BLBC    COMPONENTS_IN_PATHNAME, 36$       ; 1175
                                                52        28   A6  9E 002BA     MOVAB   40(R6), SUBVECTOR                       ; 1176
                                                50        1B   A6  9A 002BE     MOVZBL  27(NODEPTR), R0                         ; 1176
                                             20 AE                50  C0 002C2     ADDL2   R0, EXPECTED_SUBS
                                             18 AE        1B   A6  9A 002C6     MOVZBL  27(NODEPTR), 24(SP)                     ; 1176
                                                59        08   AC  D0 002CB     MOVL    SUBSCR_DESC, R9                         ; 1177
                                                57             01  CE 002CF     MNEGL   #1, J                                   ; 1187
                                                          0158  31 002D2         BRW     58$
                                  54    01    A9              08              00  ED 002D5 38$:   CMPZV   #0, #8, 1(R9), SUBSCR_INDEX  ; 1177
                                                                              10  14 002DB         BGTR    40$
                                                                        01    A9  95 002DD         TSTB    1(R9)               ; 1177
                                                                              04  12 002E0         BNEQ    39$
                                                                              53  D5 002E2         TSTL    I                   ; 1177
                                                                              07  13 002E4         BEQL    40$
                                                                        1C    AE                  01  D0 002E6 39$:   MOVL    #1, TOOFEWSUB       ; 1178
                                                                        01    A9  94 002EA         CLRB    1(R9)               ; 1178
                                                                        01    A9  95 002ED 40$:   TSTB    1(R9)               ; 1179
                                                                              23  12 002F0         BNEQ    43$
                                                                              51  1B   A6  9A 002F2 MOVZBL  27(NODEPTR), R1    ; 1179
                                                                              50  01  CE 002F6     MNEGL   #1, J
                                                                              16  11 002F9         BRB     42$
                                                                        5A                  54  C1 002FB 41$:   ADDL3   J, SUBSCR_INDEX, R10  ; 1179
                                                                              5A  0C  C4 002FF     MULL2   #12, R10                ; 1179
                                                                        02 AA49             01  88 00302     BISB2   #1, 2(R10)[R9]
                                                                                  6A49  9F 00307     PUSHAB  (R10)[R9]           ; 1179
                                                                              9E 00020000  8F  C8 0030A     BISL2   #131072, @(SP)+
                                                                        E6                  50  F3 00311 42$:   AOBLEQ  R1, J, 41$      ; 1179
                                                                     08 AE                  57  14  C5 00315 43$:   MULL3   #20, J, 8(SP)   ; 1185
                                                                        50  08   AE              08  C1 0031A     ADDL3   #8, 8(SP), R0   ; 1185
                                                                     0C AE                  52  50  C1 0031F     ADDL3   R0, SUBVECTOR, 12(SP)
                                                                        5A                  54  0C  C5 00324     MULL3   #12, SUBSCR_INDEX, R10  ; 1180
```

```
                                03       02 AA49              00  E0 00328          BBS     #0, 2(R10)[R9], 44$                                 ;
                                                           0090  31 0032E          BRW     53$                                                 ;
                                                             53  D5 00331 44$:     TSTL    I                                                   ; 1181
                                                             1F  13 00333          BEQL    47$                                                 ;
                                                          01 A9  95 00335          TSTB    1(R9)                                               ; 1181
                                                             1A  13 00338          BEQL    47$                                                 ;
                                08             6A49           11  E1 0033A          BBC     #17, (R10)[R9], 45$                                 ; 1181
                                                      00028F68 8F  DD 0033F          PUSHL   #167784                                             ; 1181
                                                             06  11 00345          BRB     46$                                                 ;
                                                      00028F60 8F  DD 00347 45$:     PUSHL   #167776                                             ; 1182
                                   000000000G  00              01  FB 0034D 46$:     CALLS   #1, LIB$SIGNAL                                      ;
                                25       0A  A6              03  E0 00354 47$:     BBS     #3, 10(NODEPTR), 50$                                ; 1182
                                         0A  A6              08  88 00359          BISB2   #8, 10(NODEPTR)                                     ; 1182
                                             50              01  CE 0035D          MNEGL   #1, K                                               ; 1182
                                                             18  11 00360          BRB     49$                                                 ;
                                51           50             14  C5 00362 48$:     MULL3   #20, K, R1                                          ;
                                                          08 A142  9F 00366          PUSHAB  8(R1)[SUBVECTOR]                                    ; 1183
                                                          6142     9F 0036A          PUSHAB  (R1)[SUBVECTOR]
                                             9E                9E  D0 0036D          MOVL    @(SP)+, @(SP)+                                      ;
                                                          0C A142  9F 00370          PUSHAB  12(R1)[SUBVECTOR]                                   ; 1183
                                                          6142     9F 00374          PUSHAB  (R1)[SUBVECTOR]
                                             9E                9E  D0 00377          MOVL    @(SP)+, @(SP)+                                      ;
                                E4           50                57  F2 0037A 49$:     AOBLSS  J, K, 48$                                           ; 1182
                                34             6A49            11  E0 0037E 50$:     BBS     #17, (R10)[R9], 52$                                 ; 1183
                                                          08 AA49  9F 00383          PUSHAB  8(R10)[R9]                                          ; 1184
                                                          04 AA49  9F 00387          PUSHAB  4(R10)[R9]
                                             9E                9E  D1 0038B          CMPL    @(SP)+, @(SP)+                                      ;
                                                               0D  15 0038E          BLEQ    51$                                                 ;
                                                      00028F08 8F  DD 00390          PUSHL   #167688                                             ; 1184
                                   000000000G  00              01  FB 00396          CALLS   #1, LIB$SIGNAL                                      ;
                                50           57                14  C5 0039D 51$:     MULL3   #20, J, R0                                          ; 1184
                                                             08 A042  9F 003A1          PUSHAB  8(R0)[SUBVECTOR]                                    ; 1184
                                                             04 AA49  9F 003A5          PUSHAB  4(R10)[R9]
                                             9E                9E  D0 003A9          MOVL    @(SP)+, @(SP)+                                      ;
                                                             0C A042  9F 003AC          PUSHAB  12(R0)[SUBVECTOR]                                   ; 1185
                                                             08 AA49  9F 003B0          PUSHAB  8(R10)[R9]
                                             9E                9E  D0 003B4          MOVL    @(SP)+, @(SP)+                                      ;
                                                             08 BE42  9F 003B7 52$:     PUSHAB  @8(SP)[SUBVECTOR]                                   ; 1185
                                             9E              10 BE  D0 003BB          MOVL    @16(SP), @(SP)+                                     ;
                                                               47  11 003BF          BRB     56$                                                 ; 1185
                                                             04 AA49  9F 003C1 53$:     PUSHAB  4(R10)[R9]                                          ; 1186
                                             50                9E  D0 003C5          MOVL    @(SP)+, VAL                                         ;
                                         0C  BE                50  D1 003C8          CMPL    VAL, @12(SP)                                        ; 1186
                                                               0F  19 003CC          BLSS    54$                                                 ;
                                51       08  AE                0C  C1 003CE          ADDL3   #12, 8(SP), R1                                      ; 1186
                                6E           52                51  C1 003D3          ADDL3   R1, SUBVECTOR, (SP)                                 ;
                                         00  BE                50  D1 003D7          CMPL    VAL, @0(SP)                                         ;
                                                               20  15 003DB          BLEQ    55$                                                 ;
                                7E       08  AE                0C  C1 003DD 54$:     ADDL3   #12, 8(SP), -(SP)                                   ; 1187
                                                          9E42     9F 003E2          PUSHAB  @(SP)+[SUBVECTOR]                                   ;
                                                             9E    DD 003E5          PUSHL   @(SP)+                                              ;
                                                          10 BE    DD 003E7          PUSHL   @16(SP)                                             ; 1186
                                                             50    DD 003EA          PUSHL   VAL                                                 ; 1186
                                                             54    DD 003EC          PUSHL   SUBSCR_INDEX                                        ;
                                                             04    DD 003EE          PUSHL   #4                                                  ;
                                                      0002868B 8F  DD 003F0          PUSHL   #165515                                             ;
                                   000000000G  00              06  FB 003F6          CALLS   #6, LIB$SIGNAL                                      ;
```

```
                                08 BE42  9F 003FD 55$:    PUSHAB  @8(SP)[SUBVECTOR]                       ; 1187
                                04 AA49  9F 00401          PUSHAB  4(R10)[R9]
                             9E          9E  D0 00405          MOVL    @(SP)+, @(SP)+
                1E        0A    A6       03  E1 00408 56$:    BBC     #3, 10(NODEPTR), 57$                ; 1187
                18        02 AA49        00  E0 0040D          BBS     #0, 2(R10)[R9], 57$                 ; 1187
                                08 BE42  9F 00413          PUSHAB  @8(SP)[SUBVECTOR]                       ; 1188
                          0C    BE       9E  D0 00417          MOVL    @(SP)+, @12(SP)
                50        08    AE       0C  C1 00418          ADDL3   #12, 8(SP), R0                      ; 1188
                51              52       50  C1 00420          ADDL3   R0, SUBVECTOR, R1
                                08 BE42  9F 00424          PUSHAB  @8(SP)[SUBVECTOR]
                              61         9E  D0 00428          MOVL    @(SP)+, (R1)
                                         54  D6 0042B 57$:    INCL    SUBSCR_INDEX                        ; 1188
                02            57    18   AE  F2 0042D 58$:    AOBLSS  24(SP), J, 59$                      ; 1176
                                         03  11 00432          BRB     60$
                                      FE9E  31 00434 59$:    BRW     38$
                          1F    A6    1B   A6  90 00437 60$:    MOVB    27(NODEPTR), 31(NODEPTR)            ; 1189
                2D        0A    A6         03  E0 0043C          BBS     #3, 10(NODEPTR), 61$                ; 1189
                          0A    A6         01  88 00441          BISB2   #1, 10(NODEPTR)                     ; 1190
                          38    AE    24   A6  D0 00445          MOVL    36(NODEPTR), TYPEID                 ; 1190
                                38    AE   DD  0044A          PUSHL   TYPEID                              ; 1190
                    00000000G   00         01  FB 0044D          CALLS   #1, DBG$STA_TYPEFCODE
                          3C    AE         50  D0 00454          MOVL    R0, FCODE
                                         7E  D4 00458          CLRL    -(SP)                               ; 1190
                                3C    AE   DD  0045A          PUSHL   TYPEID                              ; 1190
                                44    AE   DD  0045D          PUSHL   FCODE
                              7E         06  7D 00460          MOVQ    #6, -(SP)                           ; 1190
                                         5B  DD 00463          PUSHL   PRIMPTR
                        C26B    CF       06  FB 00465          CALLS   #6, DBG$BUILD_PRIMARY_SUBNODE
                              56    18   AB  D0 0046A          MOVL    24(PRIMPTR), NODEPTR                ; 1190
                              07    3C   AE  D1 0046E 61$:    CMPL    FCODE, #7                           ; 1191
                                         42  12 00472          BNEQ    64$
                              3B 00000000'   EF  E9 00474          BLBC    COMPONENTS_IN_PATHNAME, 64$         ; 1191
                                         53  D5 0047B          TSTL    I                                   ; 1192
                                         37  13 0047D          BEQL    64$
                          0A    A6         01  88 0047F          BISB2   #1, 10(NODEPTR)                     ; 1193
                          14    AE    48 AE43  D0 00483          MOVL    SYMID_VECT-4[I], SYMID1             ; 1193
                              51    38   AE  D0 00489          MOVL    TYPEID, R1                          ; 1193
                              58    2C   A1  9E 0048D          MOVAB   44(R1), TYPCOMPLST
                              50         01  CE 00491          MNEGL   #1, J                               ; 1194
                                         1B  11 00494          BRB     63$
                          10    AE       6840  D0 00496 62$:    MOVL    (TYPCOMPLST)[J], SYMID2             ; 1193
                    57    10    AE         0C  C1 0049B          ADDL3   #12, SYMID2, R7                     ; 1194
                    59    14    AE         0C  C1 004A0          ADDL3   #12, SYMID1, R9
                              67         69  D1 004A5          CMPL    (R9), (R7)
                                         07  12 004A8          BNEQ    63$
                18  A6              50         01  A1 004AA          ADDW3   #1, J, 24(NODEPTR)              ; 1194
                                         05  11 004AF          BRB     64$                                 ; 1194
                    E0              50    28   A1  F2 004B1 63$:    AOBLSS  40(R1), J, 62$                  ; 1193
                                    02         53  F4 004B6 64$:    SOBGEQ  I, 65$                          ; 1172
                                         03  11 004B9          BRB     66$
                                      FD84  31 004BB 65$:    BRW     32$
                                         55  D4 004BE 66$:    CLRL    PICKED_UP_SUBSTRING                 ; 1196
                                57    08   AC  D0 004C0          MOVL    SUBSCR_DESC, R7                     ; 1196
                                50    01   A7  9A 004C4          MOVZBL  1(R7), R0
                                         50  D7 004C8          DECL    R0
                                50         54  D1 004CA          CMPL    SUBSCR_INDEX, R0
                                         03  13 004CD          BEQL    68$
```

```
                          0137  31 004CF 67$:   BRW     80$                              ;
5A            54            0C  C5 004D2 68$:   MULL3   #12, SUBSCR_INDEX, R10           ; 1196
F3       02 AA47            00  E1 004D6        BBC     #0, 2(R10)[R7], 67$              ;
5A            54            0C  C5 004DC        MULL3   #12, SUBSCR_INDEX, R10           ; 1196
EA          6A47            12  E1 004E0        BBC     #18, (R10)[R7], 67$              ;
5A            54            0C  C5 004E5        MULL3   #12, SUBSCR_INDEX, R10           ; 1196
E1          6A47            11  E0 004E9        BBS     #17, (R10)[R7], 67$              ;
                    44      AE  D4 004EE        CLRL    DTYPE                            ; 1196
              06    2C      AE  D1 004F1        CMPL    KIND, #6                         ; 1197
                            78  12 004F5        BNEQ    73$                              ;
              50    3C      AE  D0 004F7        MOVL    FCODE, R0                        ; 1197
              02            50  D1 004FB        CMPL    R0, #2                           ; 1197
                            17  12 004FE        BNEQ    69$                              ;
                    40      AE  9F 00500        PUSHAB  BITSIZE                          ; 1197
                    48      AE  9F 00503        PUSHAB  DTYPE                            ;
                    40      AE  DD 00506        PUSHL   TYPEID                           ;
     00000000G      00      03  FB 00509        CALLS   #3, DBG$STA_TYP_ATOMIC           ;
52        40        AE      08  C7 00510        DIVL3   #8, BITSIZE, LEN                 ; 1197
                            33  11 00515        BRB     72$                              ; 1197
              03            50  D1 00517 69$:   CMPL    R0, #3                           ; 1198
                            1B  12 0051A        BNEQ    70$                              ;
                    48      AE  9F 0051C        PUSHAB  DESCR                            ; 1198
                    3C      AE  DD 0051F        PUSHL   TYPEID                           ;
     00000000G      00      02  FB 00522        CALLS   #2, DBG$STA_TYP_DESCR            ;
              50    48      AE  D0 00529        MOVL    DESCR, R0                        ; 1198
        44    AE    02      A0  9A 0052D        MOVZBL  2(R0), DTYPE                     ;
              52            60  3C 00532        MOVZWL  (R0), LEN                        ; 1198
                            13  11 00535        BRB     72$                              ; 1197
              05            50  D1 00537 70$:   CMPL    R0, #5                           ; 1198
                            05  13 0053A        BEQL    71$                              ;
              07            50  D1 0053C        CMPL    R0, #7                           ;
                            09  12 0053F        BNEQ    72$                              ;
        44    AE            0E  D0 00541 71$:   MOVL    #14, DTYPE                       ; 1198
              52  2710      8F  3C 00545        MOVZWL  #10000, LEN                      ; 1198
              50    44      AE  D0 0054A 72$:   MOVL    DTYPE, R0                        ; 1199
              0E            50  D1 0054E        CMPL    R0, #14                          ;
                            21  13 00551        BEQL    74$                              ;
              0F            50  D1 00553        CMPL    R0, #15                          ; 1199
                            1C  13 00556        BEQL    74$                              ;
              10            50  D1 00558        CMPL    R0, #16                          ; 1199
                            17  13 0055B        BEQL    74$                              ;
              11            50  D1 0055D        CMPL    R0, #17                          ; 1199
                            12  13 00560        BEQL    74$                              ;
              12            50  D1 00562        CMPL    R0, #18                          ; 1199
                            0D  13 00565        BEQL    74$                              ;
              13            50  D1 00567        CMPL    R0, #19                          ; 1199
                            08  13 0056A        BEQL    74$                              ;
              14            50  D1 0056C        CMPL    R0, #20                          ; 1199
                            03  13 0056F 73$:   BEQL    74$                              ;
                          0088  31 00571        BRW     79$                              ;
        04    AB            01  8A 00574 74$:   BICB2   #1, 4(PRIMPTR)                   ; 1200
        04    AB            02  88 00578        BISB2   #2, 4(PRIMPTR)                   ; 1200
5A            54            0C  C5 0057C        MULL3   #12, SUBSCR_INDEX, R10           ; 1200
               04 AA47      9F  9F 00580        PUSHAB  4(R10)[R7]                       ;
              53            9E  D0 00584        MOVL    @(SP)+, R3                       ;
     0007FFFF   8F          53  D1 00587        CMPL    R3, #524287                      ;
                            11  15 0058E        BLEQ    75$                              ;
```

```
                                53   DD 00590          PUSHL   R3                                      ; 1201
                                01   DD 00592          PUSHL   #1                                      ; 1201
                    000280F0    8F   DD 00594          PUSHL   #164080                                 ;
00000000G   00                  03   FB 00596          CALLS   #3, LIB$SIGNAL                          ;
         10 AB                  53   B0 005A1 75$:      MOVW    R3, 16(PRIMPTR)                         ; 1201
                      08 AA47   9F   9F 005A5          PUSHAB  8(R10)[R7]                              ; 1201
                         5A     9E   D0 005A9          MOVL    @(SP)+, R10                             ;
                         50  FF AA43  9E 005AC          MOVAB   -1(R10)[R3], R0                         ; 1201
                         52     50   D1 005B1          CMPL    R0, LEN                                 ; 1201
                                04   14 005B4          BGTR    76$                                     ;
                                53   D5 005B6          TSTL    R3                                      ; 1201
                                15   14 005B8          BGTR    77$                                     ;
                                52   DD 005BA 76$:      PUSHL   LEN                                     ; 1202
                          0408  8F   BB 005BC          PUSHR   #^M<R3,R10>                             ; 1202
                                03   DD 005C0          PUSHL   #3                                      ; 1202
                    000280D8    8F   DD 005C2          PUSHL   #164056                                 ;
00000000G   00                  05   FB 005C8          CALLS   #5, LIB$SIGNAL                          ;
00007FFF    8F                  5A   D1 005CF 77$:      CMPL    R10, #32767                             ; 1202
                                11   15 005D6          BLEQ    78$                                     ;
                                5A   DD 005D8          PUSHL   R10                                     ; 1203
                                01   DD 005DA          PUSHL   #1                                      ; 1202
                    000280F8    8F   DD 005DC          PUSHL   #164088                                 ;
00000000G   00                  03   FB 005E2          CALLS   #3, LIB$SIGNAL                          ;
         12 AB                  5A   B0 005E9 78$:      MOVW    R10, 18(PRIMPTR)                        ; 1203
                         56     18 AB   D0 005ED          MOVL    24(PRIMPTR), NODEPTR                    ; 1203
                      14 A6             01   CE 005F1          MNEGL   #1, 20(NODEPTR)                 ; 1203
                         55             01   D0 005F5          MOVL    #1, PICKED_UP_SUBSTRING         ; 1203
                                        54   D6 005F8          INCL    SUBSCR_INDEX                    ; 1203
                                        0D   11 005FA          BRB     80$                             ; 1199
                              00028F70  8F   DD 005FC 79$:      PUSHL   #167792                         ; 1205
                    00000000G   00      01   FB 00602          CALLS   #1, LIB$SIGNAL                  ;
                                12   1C AE   E9 00609 80$:      BLBC    TOOFEWSUB, 81$                  ; 1205
                                     20 AE   DD 0060D          PUSHL   EXPECTED_SUBS                   ; 1206
                                        01   DD 00610          PUSHL   #1                              ;
                              00028EA0  8F   DD 00612          PUSHL   #167584                         ;
                    00000000G   00      03   FB 00618          CALLS   #3, LIB$SIGNAL                  ;
      54      01    A7                08   00   ED 0061F 81$:      CMPZV   #0, #8, 1(R7), SUBSCR_INDEX     ; 1206
                                           23   15 00625          BLEQ    84$                             ;
                      7E                54   55   C3 00627          SUBL3   PICKED_UP_SUBSTRING, SUBSCR_INDEX, - SP)  ; 1206
                                           01   DD 0062B          PUSHL   #1                              ;
                                    00028EB0  8F   DD 0062D          PUSHL   #167600                         ;
                                              0E   11 00633          BRB     83$                             ;
                                    00000000' EF   9F 00635 82$:      PUSHAB  P.AYS                           ; 1207
                                              01   DD 0063B          PUSHL   #1                              ;
                                    00028362  8F   DD 0063D          PUSHL   #164706                         ;
                          00000000G   00      03   FB 00643 83$:      CALLS   #3, LIB$SIGNAL                  ;
                                50            5B   D0 0064A 84$:      MOVL    PRIMPTR, R0                     ; 1207
                                              04   0064D          RET                                     ; 1207

; Routine Size:  1614 bytes,    Routine Base:  DBG$CODE + 3997
```

```
;11992     12079  1     ROUTINE RESOLVE_COMPONENT (TYPEID, COMP_LIST, SYMID, PRIMPTR, COMPNAME) =
;11993     12080  1     !
;11994     12081  1     ! FUNCTION
;11995     12082  1     !       This routine is called from GET_RECORD_COMPONENT to resolve
;11996     12083  1     !       possible ambiguities where the user has specified X.Y and
;11997     12084  1     !       there is more than one record component named Y.
;11998     12085  1     !
;11999     12086  1     !       This situation arises in C, where membership checking is not
;12000     12087  1     !       enforced. Thus in C you can say X.Y even though Y is not a
;12001     12088  1     !       component in the record given by X. If we find more than one
;12002     12089  1     !       symid for Y, there is a possible ambiguity (although if all
;12003     12090  1     !       of the symids are record components with the same type and
;12004     12091  1     !       offset, then it is OK).
;12005     12092  1     !
;12006     12093  1     !       A different situation arises in BASIC. Here, we allow A::C to
;12007     12094  1     !       be an abbreviation for A::B::C (incomplete data qualification).
;12008     12095  1     !       In this case, the INCOMPLETE_QUAL flag is lit. What we do here
;12009     12096  1     !       is call a routine which chases upscope pointers to determine
;12010     12097  1     !       whether we can get to the given TYPEID from the component SYMID.
;12011     12098  1     !
;12012     12099  1     ! INPUTS
;12013     12100  1     !       TYPEID          - typeid for the record
;12014     12101  1     !       COMP_LIST       - list of symids which are record components having
;12015     12102  1     !                         the same name. This list is in the form of
;12016     12103  1     !                         a vector of longword, with the first longword
;12017     12104  1     !                         being the count.
;12018     12105  1     !       SYMID           - address in which to leave the resolved symid,
;12019     12106  1     !                         if one is determined.
;12020     12107  1     !       PRIMPTR         - pointer to the input Primary
;12021     12108  1     !       COMPNAME        - name of the component
;12022     12109  1     !
;12023     12110  1     ! OUTPUTS
;12024     12111  1     !       The value TRUE is returned if a unique symid was determined;
;12025     12112  1     !       FALSE otherwise. If TRUE is returned then the output parameter
;12026     12113  1     !       SYMID is filled in.
;12027     12114  1     !
;12028     12115  2         BEGIN
;12029     12116  2         MAP
;12030     12117  2             COMP_LIST: REF VECTOR[];
;12031     12118  2         LOCAL
;12032     12119  2             FOUND;
;12033     12120  2
;12034     12121  2         DBG$GL_CURRENT_PRIMARY = .PRIMPTR;
;12035     12122  2
;12036     12123  2         ! If incomplete data qualification is allowed in this language
;12037     12124  2         ! (for example A::C in place of A::B::C) then we search the list
;12038     12125  2         ! of candidate components to see if there is a unique one in the
;12039     12126  2         ! given record. For example, if the user says A::C, and there are
;12040     12127  2         ! several components C, but only one belongs in record A, then
;12041     12128  2         ! that is the one we want.
;12042     12129  2         !
;12043     12130  2         ! This is the code path taken for language BASIC.
;12044     12131  2         !
;12045     12132  2         IF .INCOMPLETE_QUAL
;12046     12133  2         THEN
;12047     12134  3             BEGIN
;12048     12135  3             FOUND = FALSE;
```

```
;12049          12136  3          INCR I FROM 1 TO .COMP_LIST[0] DO
;12050          12137  3
;12051          12138  3              ! The CHECK_UPSCOPE routine determines whether the given
;12052          12139  3              ! record component belongs in the record given by TYPEID.
;12053          12140  3              ! If it does, but there are intervening record components,
;12054          12141  3              ! then the Primary must be modified to include the component
;12055          12142  3              ! selection for these intervening components.
;12056          12143  3              !
;12057          12144  3              IF CHECK_UPSCOPE(.COMP_LIST[.I], .TYPEID, .PRIMPTR, 0)
;12058          12145  3              THEN
;12059          12146  3                  IF .FOUND
;12060          12147  3                  THEN
;12061          12148  3
;12062          12149  3                      ! Not unique.
;12063          12150  3                      !
;12064          12151  3                      RETURN FALSE
;12065          12152  3                  ELSE
;12066          12153  4                      BEGIN
;12067          12154  4
;12068          12155  4                      ! We found one.
;12069          12156  4                      !
;12070          12157  4                      .SYMID = .COMP_LIST[.I];
;12071          12158  4                      FOUND = TRUE;
;12072          12159  3                      END;
;12073          12160  3
;12074          12161  3          ! If we failed to find one, signal that the given component
;12075          12162  3          ! is not a field of this record. Note that we do not return FALSE
;12076          12163  3          ! here - returning FALSE indicates an ambiguous field name.
;12077          12164  3          !
;12078          12165  3              IF NOT .FOUND
;12079          12166  3              THEN
;12080          12167  3                  SIGNAL(DBG$_NOFIELD, 1, .COMPNAME);
;12081          12168  3
;12082          12169  3              ! Found a unique component - return true.
;12083          12170  3              !
;12084          12171  3              RETURN TRUE;
;12085          12172  2              END;
;12086          12173  2
;12087          12174  2
;12088          12175  2          ! This is where we end up for C: we are trying to resolve an ambiguity.
;12089          12176  2          ! Check for only one component in list: then there is no ambiguity.
;12090          12177  2          !
;12091          12178  2          IF .COMP_LIST[0] EQL 1
;12092          12179  2          THEN
;12093          12180  3              BEGIN
;12094          12181  3              .SYMID = .COMP_LIST[1];
;12095          12182  3              RETURN TRUE;
;12096          12183  2              END;
;12097          12184  2
;12098          12185  2          ! More than one component.
;12099          12186  2          !
;12100          12187  2          RETURN FALSE;
;12101          12188  1          END;
```

```
                                000C 00000 RESOLVE_COMPONENT:
                                           .WORD   Save R2,R3                                  ; 1207
00000000G   00          10   AC  D0 00002         MOVL    PRIMPTR, DBG$GL_CURRENT_PRIMARY     ; 1212
            40 00000000'     EF  E9 0000A         BLBC    INCOMPLETE_QUAL, 3$                 ; 1213
                             52  7C 00011         CLRQ    I                                   ; 1214
                             20  11 00013         BRB     2$                                  ;
                             7E  D4 00015 1$:     CLRL    -(SP)                               ;
                         10  AC  DD 00017         PUSHL   PRIMPTR                             ;
                         04  AC  DD 0001A         PUSHL   TYPEID                              ;
                         08 BC42 DD 0001D         PUSHL   @COMP_LIST[I]                       ;
                E2FB     CF      04  FB 00021     CALLS   #4, CHECK_UPSCOPE                   ;
                         0C      50  E9 00026     BLBC    R0, 2$                              ;
                         37      53  E8 00029     BLBS    FOUND, 5$                           ; 1214
                   0C    BC      08 BC42 D0 0002C MOVL    @COMP_LIST[I], @SYMID               ; 1215
                         53          01  D0 00032 MOVL    #1, FOUND                           ; 1215
DB                       52      08  BC  F3 00035 2$:     AOBLEQ  @COMP_LIST  I, 1$           ; 1214
                         22          53  E8 0003A BLBS    FOUND, 4$                           ; 1216
                             14  AC  DD 0003D         PUSHL   COMPNAME                        ; 1216
                                 01  DD 00040         PUSHL   #1                              ;
                      00028C80   8F  DD 00042         PUSHL   #167040                         ;
00000000G   00                       03  FB 00048     CALLS   #3, LIB$SIGNAL                  ;
                                     0E  11 0004F     BRB     4$                              ; 1217
                         50      08  AC  D0 00051 3$:     MOVL    COMP_LIST, R0               ; 1217
                         01          60  D1 00055 CMPL    (R0), #1                            ;
                                     09  12 00058 BNEQ    5$                                  ;
                   0C    BC      04  A0  D0 0005A MOVL    4(R0), @SYMID                       ; 1218
                         50          01  D0 0005F 4$:     MOVL    #1, R0                      ; 1218
                                         04 00062         RET                                 ;
                                     50  D4 00063 5$:     CLRL    R0                          ; 1218
                                         04 00065         RET                                 ;

; Routine Size:  102 bytes,    Routine Base:  DBG$CODE + 3FE5
```

```
;12103      12189  1     ROUTINE SAVE_SUBSCRIPTS(PATHDESC, SUBSCR_DESC): NOVALUE =
;12104      12190  1     !
;12105      12191  1     ! FUNCTION
;12106      12192  1     !       When parsing PLI or BASIC Primaries we want to save away the subscripts
;12107      12193  1     !       that we see along the way. This routine picks up subscripts and
;12108      12194  1     !       saves them into the SUBSCR_DESC data structure.
;12109      12195  1     !
;12110      12196  1     ! INPUTS
;12111      12197  1     !       PATHDESC        -       A pointer to the pathname descriptor for
;12112      12198  1     !                               the pathname we have parsed so far.
;12113      12199  1     !       SUBSCR_DESC     -       A data structure containing the subscript
;12114      12200  1     !                               values.
;12115      12201  1     !
;12116      12202  1     ! OUTPUTS
;12117      12203  1     !
;12118      12204  2         BEGIN
;12119      12205  2         MAP
;12120      12206  2             PATHDESC: REF PTH$PATHNAME,
;12121      12207  2             SUBSCR_DESC: REF SUBSCR$DESC;
;12122      12208  2
;12123      12209  2         LOCAL
;12124      12210  2             DECLTYPE: REF DBG$VALDESC,      ! Pointer to Value Descriptor for
;12125      12211  2                                             !       declared subscript data type
;12126      12212  2             LA_PTR: REF VECTOR[,BYTE],      ! Lookahead pointer into input
;12127      12213  2             LOW_RANGE_VAL,                  ! Low value of a subscript range
;12128      12214  2             PATH_INDEX,                     ! Count of pathname components
;12129      12215  2             SAVED_RADIX,                    ! Temporarily saved expression radix
;12130      12216  2             SUBSCR_COUNT,                   ! Actual subscript count in input line
;12131      12217  2             THIS_SUBSCR_IS_RANGE,           ! Flag set if the current subscript is
;12132      12218  2                                             !       given as a subscript range
;12133      12219  2             TOKEN,                          ! Lexical Token
;12134      12220  2             VALADDR: REF VECTOR[,LONG],     ! Pointer to integer subscript value
;12135      12221  2             VALPTR: REF DBG$VALDESC;        ! Pointer to subscript Value Descriptor
;12136      12222  2
;12137      12223  2
;12138      12224  2
;12139      12225  2         ! Note how many subscripts we have previously picked up (i.e., in earlier
;12140      12226  2         ! calls to this routine, while parsing this same expression). In PL/I,
;12141      12227  2         ! for example, the subscripts may arrive in separate pieces, and do not
;12142      12228  2         ! necessarily have to be associated with the "right" component, e.g.,
;12143      12229  2         ! X(1,2).Y(3).Z(4,5)
;12144      12230  2         ! This routine also picks up substring references in COBOL. Because of this,
;12145      12231  2         ! we set a marker to indicate where we were in SUBSCR_DESC when we entered
;12146      12232  2         ! this routine. E.g., in the COBOL expression X(1,2,3)(1:5) we set a
;12147      12233  2         ! marker to indicate that the 1:5 came in a separate set of parenthesis.
;12148      12234  2         ! It would be illegal otherwise, so this marker can be used later to
;12149      12235  2         ! decide whether to signal an error.
;12150      12236  2         !
;12151      12237  2         SUBSCR_COUNT = .SUBSCR_DESC[0, SUBSCR$B_SUBCNT];
;12152      12238  2         SUBSCR_DESC[.SUBSCR_COUNT, SUBSCR$V_MARKER] = TRUE;
;12153      12239  2
;12154      12240  2
;12155      12241  2         ! Loop through the subscript expressions for this array reference.  Each
;12156      12242  2         ! subscript is parsed, evaluated, and converted to integer. Its value
;12157      12243  2         ! is then stored in the SUBSCR_DESC data structure.
;12158      12244  2         !
;12159      12245  2         PATH_INDEX = .PATHDESC[PTH$B_TOTCNT];
```

```
;12160      12246  2          THIS_SUBSCR_IS_RANGE = FALSE;
;12161      12247  2          TERMINATOR_CODE = TOKEN$K_TERM_COMMA;
;12162      12248  2          WHILE .TERMINATOR_CODE NEQ TOKEN$K_TERM_CLOSE DO
;12163      12249  3              BEGIN
;12164      12250  3
;12165      12251  3
;12166      12252  3              ! Look for the asterisk. X(*) is the same as X(lower:upper).
;12167      12253  3              ! If we find the asterisk then advance the character pointer beyond
;12168      12254  3              ! the asterisk and also increment the subscript count.
;12169      12255  3              !
;12170      12256  3              LA_PTR = .CHARPTR;
;12171      12257  3              WHILE .LA_PTR[0] EQL ' ' DO LA_PTR = .LA_PTR + 1;
;12172      12258  3              IF .LA_PTR[0] EQL '*'
;12173      12259  3              THEN
;12174      12260  4                  BEGIN
;12175      12261  4                  CHARPTR = .LA_PTR + 1;
;12176      12262  4
;12177      12263  4
;12178      12264  4                      ! Call the Lexical Scanner to take us past the ',' or
;12179      12265  4                      ! or ']' or ')'. This will set TERMINATOR_CODE to the
;12180      12266  4                      ! terminator that is seen. If we do not see a terminator
;12181      12267  4                      ! then signal a syntax error. Also signal an error if
;12182      12268  4                      ! ':' was the terminator.
;12183      12269  4                      !
;12184      12270  4                      TOKEN = DBG$LEXICAL_SCANNER (FALSE, FALSE,
;12185      12271  4                                  .SUBSCRIPT_TERM_TBL, 0);
;12186      12272  4                      IF .TOKEN NEQ TERMINATOR_TOKEN
;12187      12273  4                      THEN
;12188      12274  5                          BEGIN
;12189      12275  5                          LOCAL
;12190      12276  5                              ASCIC_STRING: VECTOR[2,BYTE];
;12191      12277  5                          ASCIC_STRING[0] = 1;
;12192      12278  5                          ASCIC_STRING[1] = .CHARPTR[0];
;12193      12279  5                          SIGNAL(DBG$_SYNERREXPR, 1, ASCIC_STRING);
;12194      12280  4                          END;
;12195      12281  4                      IF .TERMINATOR_CODE EQL TOKEN$K_TERM_COLON
;12196      12282  4                      THEN
;12197      12283  4                          SIGNAL (DBG$_INVRANSPEC);
;12198      12284  4                      IF .TERMINATOR_CODE EQL TOKEN$K_TERM_NONE
;12199      12285  4                      THEN
;12200      12286  4                          SIGNAL(DBG$_MISCLOSUB);
;12201      12287  4                      CHARPTR = .CHARPTR + .TERMINATOR_LENGTH;
;12202      12288  4
;12203      12289  4
;12204      12290  4                      SUBSCR_DESC[.SUBSCR_COUNT, SUBSCR$B_PATH_INDEX] = .PATH_INDEX;
;12205      12291  4                      SUBSCR_DESC[.SUBSCR_COUNT, SUBSCR$V_ASTER] = TRUE;
;12206      12292  4                      SUBSCR_DESC[.SUBSCR_COUNT, SUBSCR$V_RANGE] = TRUE;
;12207      12293  4                      SUBSCR_COUNT = .SUBSCR_COUNT + 1;
;12208      12294  4                      END
;12209      12295  4
;12210      12296  3                  ELSE
;12211      12297  4                      BEGIN
;12212      12298  4
;12213      12299  4
;12214      12300  4                      ! Call the expression parser to pick up the next subscript expression
;12215      12301  4                      ! and its value.  Note that we set the radix to decimal over this call
;12216      12302  4                      ! and then restore it.  Also note that the Expression Parser sets
```

```
;12217          12303  4                ! TERMINATOR_CODE and TERMINATOR_LENGTH as a side-effect.
;12218          12304  4                !
;12219          12305  4                SAVED_RADIX = .EXPRESSION_RADIX;
;12220          12306  4                EXPRESSION_RADIX = DBG$K_DECIMAL;
;12221          12307  4                VALPTR = DBG$EXPRESSION_PARSER (FALSE, .SUBSCRIPT_TERM_TBL);
;12222          12308  4                EXPRESSION_RADIX = .SAVED_RADIX;
;12223          12309  4
;12224          12310  4
;12225          12311  4                ! Check the terminator code.  If there was no terminator (the input
;12226          12312  4                ! line just ended), signal an error.  Otherwise we got a comma or clos-
;12227          12313  4                ! ing subscript parenthesis and we increment CHARPTR to get past it.
;12228          12314  4                !
;12229          12315  4                IF .TERMINATOR_CODE EQL TOKEN$K_TERM_NONE THEN SIGNAL(DBG$_MISCLOSUB);
;12230          12316  4                CHARPTR = .CHARPTR + .TERMINATOR_LENGTH;
;12231          12317  4
;12232          12318  4
;12233          12319  4                ! We now need to convert the subscript to one of the appropriate
;12234          12320  4                ! dtype. We need to set up a target descriptor for the conversion
;12235          12321  4                ! routine. We allocate a skeleton descriptor and fill in some of
;12236          12322  4                ! the fields.
;12237          12323  4                !
;12238          12324  4                DECLTYPE = DBG$MAKE_SKELETON_DESC(DBG$K_VALUE_DESC, 4);
;12239          12325  4                DECLTYPE[DBG$B_DHDR_KIND] = RST$K_DATA;
;12240          12326  4                DECLTYPE[DBG$B_DHDR_FCODE] = RST$R_TYPE_ATOMIC;
;12241          12327  4                DECLTYPE[DBG$B_VALUE_CLASS] = DSC$K_CLASS_S;
;12242          12328  4                DECLTYPE[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_L;
;12243          12329  4                DECLTYPE[DBG$W_VALUE_LENGTH] = 4;
;12244          12330  4                DECLTYPE[DBG$L_VALUE_POINTER] = DECLTYPE[DBG$A_VALUE_ADDRESS];
;12245          12331  4
;12246          12332  4
;12247          12333  4                ! Finally call the conversion routine. This routine checks that
;12248          12334  4                ! the conversion is legal before doing it.
;12249          12335  4                !
;12250          12336  4                VALPTR = DBG$EVAL_LANG_OPERATOR(DBG$GL_CONVERT_TOKEN, .VALPTR, .DECLTYPE);
;12251          12337  4                VALADDR = .VALPTR[DBG$L_VALUE_POINTER];
;12252          12338  4
;12253          12339  4
;12254          12340  4                ! If the terminator at the end of this subscript expression was a colon
;12255          12341  4                ! we have a subscript range (for example, "ARR(1:5,2)").  We thus set
;12256          12342  4                ! the subscript-range flag and save the low value of the range, i.e.
;12257          12343  4                ! the value we just picked up.  If this is the first range in the
;12258          12344  4                ! subscript list, we also turn all previous subscripts into ranges by
;12259          12345  4                ! setting the lower and upper bound for each such subscript to the
;12260          12346  4                ! corresponding subscript value.  This in effect defines a new array
;12261          12347  4                ! which constitutes a "slice" of the original array.
;12262          12348  4                !
;12263          12349  4                IF .TERMINATOR_CODE EQL TOKEN$K_TERM_COLON
;12264          12350  4                THEN
;12265          12351  5                    BEGIN
;12266          12352  5                    IF .THIS_SUBSCR_IS_RANGE THEN SIGNAL(DBG$_INVRANSPEC);
;12267          12353  5                    THIS_SUBSCR_IS_RANGE = TRUE;
;12268          12354  5                    LOW_RANGE_VAL = .VALADDR[0];
;12269          12355  5                    END
;12270          12356  5
;12271          12357  5
;12272          12358  5                ! The terminator was not a colon, so we now have the full subscript
;12273          12359  5                ! specification.  Fill the subscript value into the Array Sub-Node's
```

```
;12274     12360  5          ! subscript vector.  Set up the bounds for an array "slice" if this
;12275     12361  5          ! or any previous subscript specification in this array reference
;12276     12362  5          ! consisted of a subscript range.  Also bump the subscript count.
;12277     12363  5          !
;12278     12364  4          ELSE
;12279     12365  5              BEGIN
;12280     12366  5
;12281     12367  5
;12282     12368  5              ! If this subscript is specified as a subscript range, check that
;12283     12369  5              ! the first value in the range is not greater than the second.
;12284     12370  5              ! Also clear the subscript-is-range flag for the next subscript.
;12285     12371  5              !
;12286     12372  5              IF .THIS_SUBSCR_IS_RANGE
;12287     12373  5              THEN
;12288     12374  6                  BEGIN
;12289     12375  6                  SUBSCR_DESC[.SUBSCR_COUNT, SUBSCR$V_RANGE] = TRUE;
;12290     12376  6                  THIS_SUBSCR_IS_RANGE = FALSE;
;12291     12377  6                  END
;12292     12378  6
;12293     12379  6
;12294     12380  6              ! Otherwise, set the low range value to be the subscript value.
;12295     12381  6              !
;12296     12382  5              ELSE
;12297     12383  5                  LOW_RANGE_VAL = .VALADDR[0];
;12298     12384  5
;12299     12385  5
;12300     12386  5              ! Finally fill in the subscript value itself (the start of the
;12301     12387  5              ! range), increment the subscript count, and loop.
;12302     12388  5              !
;12303     12389  5              SUBSCR_DESC[.SUBSCR_COUNT, SUBSCR$B_PATH_INDEX] = .PATH_INDEX;
;12304     12390  5              SUBSCR_DESC[.SUBSCR_COUNT, SUBSCR$L_LBOUND] = .LOW_RANGE_VAL;
;12305     12391  5              IF .SUBSCR_DESC[.SUBSCR_COUNT, SUBSCR$V_RANGE]
;12306     12392  5              THEN
;12307     12393  5                  SUBSCR_DESC[.SUBSCR_COUNT, SUBSCR$L_UBOUND] = .VALADDR[0];
;12308     12394  5              SUBSCR_COUNT = .SUBSCR_COUNT + 1;
;12309     12395  4              END;
;12310     12396  3          END;
;12311     12397  2      END;                            ! End of WHILE loop over subscripts
;12312     12398  2
;12313     12399  2
;12314     12400  2      ! We have picked up all the subscripts within this set of subscript paren-
;12315     12401  2      ! theses.
;12316     12402  2      SUBSCR_DESC[0, SUBSCR$B_SUBCNT] = .SUBSCR_COUNT;
;12317     12403  2      RETURN;
;12318     12404  1      END;


                                    0FFC 00000 SAVE_SUBSCRIPTS:
                                                       .WORD   Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11        ; 1218
                                5E          08  C2 00002       SUBL2   #8, SP                              ;
                                53      08  AC  D0 00005       MOVL    SUBSCR_DESC, R3                     ; 1223
                                54      01  A3  9A 00009       MOVZBL  1(R3), SUBSCR_COUNT                 ;
                             50 54          0C  C5 0000D       MULL3   #12, SUBSCR_COUNT, R0               ; 1223
                                          6043  9F 00011       PUSHAB  (R0)[R3]                            ;
```

```
                9E 00040000    8F   C8 00014          BISL2   #262144, @(SP)+
                59       04    BC   9A 0001B          MOVZBL  @PATHDESC, PATH_INDEX                   ; 1224
                               58   D4 0001F          CLRL    THIS_SUBSCR_IS_RANGE                    ; 1224
00000000'       EF             01   D0 00021          MOVL    #1, TERMINATOR_CODE                     ; 1224
                02 00000000'   EF   D1 00028 1$:      CMPL    TERMINATOR_CODE, #2                     ; 1224
                               03   12 0002F          BNEQ    2$
                             0181   31 00031          BRW     17$
                56 00000000'   EF   D0 00034 2$:      MOVL    CHARPTR, LA_PTR                         ; 1225
                20             66   91 0003B 3$:      CMPB    (LA_PTR), #32                           ; 1225
                               04   12 0003E          BNEQ    4$
                               56   D6 00040          INCL    LA_PTR
                               F7   11 00042          BRB     3$
                2A             66   91 00044 4$:      CMPB    (LA_PTR), #42                           ; 1225
                               03   13 00047          BEQL    5$
                             0094   31 00049          BRW     9$
00000000'       EF       01    A6   9E 0004C 5$:      MOVAB   1(R6), CHARPTR                          ; 1226
                               7E   D4 00054          CLRL    -(SP)                                   ; 1227
                00000000'      EF   DD 00056          PUSHL   SUBSCRIPT_TERM_TBL                      ; 1227
                               7E   7C 0005C          CLRQ    -(SP)                                   ; 1227
            C9F0 CF            04   FB 0005E          CALLS   #4, DBG$LEXICAL_SCANNER
                6E             50   D0 00063          MOVL    R0, TOKEN
                50 00000000'   EF   9E 00066          MOVAB   TERMINATOR_TOKEN, R0                    ; 1227
                50             6E   D1 0006D          CMPL    TOKEN, R0
                               1E   13 00070          BEQL    6$
              04 AE            01   90 00072          MOVB    #1, ASCIC_STRING                        ; 1227
              05 AE 00000000'  FF   90 00076          MOVB    @CHARPTR, ASCIC_STRING+1                ; 1227
                       04      AE   9F 0007E          PUSHAB  ASCIC_STRING                            ; 1227
                               01   DD 00081          PUSHL   #1
                000289E2       8F   DD 00083          PUSHL   #166370
00000000G       00             03   FB 00089          CALLS   #3, LIB$SIGNAL
                03 00000000'   EF   D1 00090 6$:      CMPL    TERMINATOR_CODE, #3                     ; 1228
                               0D   12 00097          BNEQ    7$
                00028F08       8F   DD 00099          PUSHL   #167688                                 ; 1228
    00000000G   00             01   FB 0009F          CALLS   #1, LIB$SIGNAL
                00000000'      EF   D5 000A6 7$:      TSTL    TERMINATOR_CODE                         ; 1228
                               0D   12 000AC          BNEQ    8$
                00028E90       8F   DD 000AE          PUSHL   #167568                                 ; 1228
    00000000G   00             01   FB 000B4          CALLS   #1, LIB$SIGNAL
    00000000'   EF 00000000'   EF   C0 000BB 8$:      ADDL2   TERMINATOR_LENGTH, CHARPTR              ; 1228
50                  54         0C   C5 000C6          MULL3   #12, SUBSCR_COUNT, R0                   ; 1229
                  6043         59   90 000CA          MOVB    PATH_INDEX, (R0)[R3]
                             6043   9F 000CE          PUSHAB  (R0)[R3]                                ; 1229
                9E 00020000    8F   C8 000D1          BISL2   #131072, @(SP)+
              02 A043          01   88 000D8          BISB2   #1, 2(R0)[R3]                           ; 1229
                             00D0   31 000DD          BRW     15$                                     ; 1229
                5B 00000000'   EF   D0 000E0 9$:      MOVL    EXPRESSION_RADIX, SAVED_RADIX           ; 1230
    00000000'   EF             0A   D0 000E7          MOVL    #10, EXPRESSION_RADIX                   ; 1230
                00000000'      EF   DD 000EE          PUSHL   SUBSCRIPT_TERM_TBL                      ; 1230
                               7E   D4 000F4          CLRL    -(SP)
            C4B2 CF            02   FB 000F6          CALLS   #2, DBG$EXPRESSION_PARSER
                55             50   D0 000FB          MOVL    R0, VALPTR
    00000000'   EF             5B   D0 000FE          MOVL    SAVED_RADIX, EXPRESSION_RADIX           ; 1230
                00000000'      EF   D5 00105          TSTL    TERMINATOR_CODE                         ; 1231
                               0D   12 0010B          BNEQ    10$
                00028E90       8F   DD 0010D          PUSHL   #167568
    00000000G   00             01   FB 00113          CALLS   #1, LIB$SIGNAL
    00000000'   EF 00000000'   EF   C0 0011A 10$:     ADDL2   TERMINATOR_LENGTH, CHARPTR              ; 1231
```

```
                                04   DD 00125          PUSHL   #4                                      ; 1232
                          7E     7A   8F  9A 00127     MOVZBL  #122, -(SP)
                00000000G 00         02  FB 0012B     CALLS   #2, DBG$MAKE_SKELETON_DESC
                          52         50  D0 00132     MOVL    R0, DECLTYPE
                      06  A2   0602   8F  B0 00135     MOVW    #1538, 6(DECLTYPE)                      ; 1232
                      14  A2 01080004 8F  D0 0013B     MOVL    #17301508, 20(DECLTYPE)                 ; 1232
                      18  A2     20   A2  9E 00143     MOVAB   32(R2), 24(DECLTYPE)                    ; 1233
                                     52  DD 00148     PUSHL   DECLTYPE                                ; 1233
                                     55  DD 0014A     PUSHL   VALPTR
                               00000000' EF  9F 0014C     PUSHAB  DBG$GL_CONVERT_TOKEN
                00000000G 00         03  FB 00152     CALLS   #3, DBG$EVAL_LANG_OPERATOR
                          55         50  D0 00159     MOVL    R0, VALPTR
                          57     18  A5  D0 0015C     MOVL    24(VALPTR), VALADDR                     ; 1233
                          03 00000000' EF  D1 00160     CMPL    TERMINATOR_CODE, #3                     ; 1234
                                     18  12 00167     BNEQ    12$
                          0D         58  E9 00169     BLBC    THIS_SUBSCR_IS_RANGE, 11$               ; 1235
                               00028F08  8F  DD 0016C     PUSHL   #167688
                00000000G 00         01  FB 00172     CALLS   #1, LIB$SIGNAL
                          58         01  D0 00179 11$: MOVL    #1, THIS_SUBSCR_IS_RANGE                ; 1235
                          5A         67  D0 0017C     MOVL    (VALADDR), LOW_RANGE_VAL                ; 1235
                                     31  11 0017F     BRB     16$                                     ; 1234
                          0D         58  E9 00181 12$: BLBC    THIS_SUBSCR_IS_RANGE, 13$               ; 1237
                50        54         0C  C5 00184     MULL3   #12, SUBSCR_COUNT, R0                   ; 1237
                      02 A043        01  88 00188     BISB2   #1, 2(R0)[R3]
                                     58  D4 0018D     CLRL    THIS_SUBSCR_IS_RANGE                    ; 1237
                                     03  11 0018F     BRB     14$                                     ; 1237
                          5A         67  D0 00191 13$: MOVL    (VALADDR), LOW_RANGE_VAL                ; 1238
                50        54         0C  C5 00194 14$: MULL3   #12, SUBSCR_COUNT, R0                   ; 1238
                        6043         59  90 00198     MOVB    PATH_INDEX, (R0)[R3]
                      04 A043        9F  9F 0019C     PUSHAB  4(R0)[R3]                               ; 1239
                          9E         5A  D0 001A0     MOVL    LOW_RANGE_VAL, @(SP)+
                07        02 A043    00  E1 001A3     BBC     #0, 2(R0)[R3], 15$                      ; 1239
                      08 A043        9F  9F 001A9     PUSHAB  8(R0)[R3]                               ; 1239
                          9E         67  D0 001AD     MOVL    (VALADDR), @(SP)+
                                     54  D6 001B0 15$: INCL    SUBSCR_COUNT                            ; 1239
                                 FE73    31 001B2 16$: BRW     1$                                      ; 1224
                      01    A3         54  90 001B5 17$: MOVB    SUBSCR_COUNT, 1(R3)                     ; 1240
                                         04 001B9     RET                                             ; 1240

; Routine Size:  442 bytes,    Routine Base:  DBG$CODE + 404B
```

```
;12320          12405  1        ROUTINE SCAN_QUOTED_STRING(TOKENBUFFER, TOKEN_TYPE): NOVALUE =
;12321          12406  1        !
;12322          12407  1        ! FUNCTION
;12323          12408  1        !       This routine scans a quoted character string and returns the found
;12324          12409  1        !       string in Counted ASCII format to a caller-provided buffer.  It
;12325          12410  1        !       expects the OWN pointer CHARPTR to point to the quote character at
;12326          12411  1        !       the start of the character constant and it assumes that this in
;12327          12412  1        !       fact is a valid quote character.  It then scans for the closing
;12328          12413  1        !       quote character (which must be the same character as the opening
;12329          12414  1        !       quote), treating doubled up quote characters within the string as
;12330          12415  1        !       a single quoted quote character.  If the language is set to PL/I, then
;12331          12416  1        !       after finding the closing quote, it searches for a "B" or "b", to
;12332          12417  1        !       determine if the string could be a bit-string rather than a
;12333          12418  1        !       character string.  If the quote characters are not "'", or if any
;12334          12419  1        !       double quotes are encountered, then the trailing "B" is not
;12335          12420  1        !       searched for.
;12336          12421  1        !
;12337          12422  1        !       If a carriage-return (end of input line) is found before the closing
;12338          12423  1        !       quote or if the string exceeds 255 characters (the longest Counted
;12339          12424  1        !       ASCII allows), an error is signalled.  Otherwise, CHARPTR is left
;12340          12425  1        !       pointing to the first character after the closing quote and the
;12341          12426  1        !       string itself is returned as Counted ASCII to the caller's buffer.
;12342          12427  1        !
;12343          12428  1        ! INPUTS
;12344          12429  1        !       TOKENBUFFER - A pointer to the buffer in which the character string is
;12345          12430  1        !                     to be accumulated.  This buffer is expected to be 256 charac-
;12346          12431  1        !                     ters long, enough for the longest Counted ASCII string.
;12347          12432  1        !
;12348          12433  1        !       TOKEN_TYPE  - Address of where to return token's type.
;12349          12434  1        !
;12350          12435  1        ! OUTPUTS
;12351          12436  1        !       TOKENBUFFER - The quoted character string is accumulated and returned
;12352          12437  1        !                     as a Counted ASCII string to the buffer pointed to by
;12353          12438  1        !                     TOKENBUFFER.
;12354          12439  1        !
;12355          12440  1        !       TOKEN_TYPE  - Is filled in with either TOKEN$K_STRING or
;12356          12441  1        !                     TOKEN$K_BIT_STRING.
;12357          12442  1        !
;12358          12443  1
;12359          12444  2            BEGIN
;12360          12445  2
;12361          12446  2            MAP
;12362          12447  2                TOKENBUFFER: REF VECTOR[,BYTE], ! Pointer to buffer for char string
;12363          12448  2                TOKEN_TYPE : REF VECTOR[1];     ! Longword to receive token's type.
;12364          12449  2
;12365          12450  2            LOCAL
;12366          12451  2                QUOTE,                          ! Quote character which started the
;12367          12452  2                                                !      current quoted string constant
;12368          12453  2                DOUBLED_QUOTES,                 ! Flag denotes whether or not there
;12369          12454  2                                                !      exist doubled up quotes ('').
;12370          12455  2                TOKENLEN;                       ! Length of quoted string so far
;12371          12456  2
;12372          12457  2
;12373          12458  2
;12374          12459  2            ! We pick up the closing quote character (which must be the same as the
;12375          12460  2            ! opening quote character) and then scan for the end of the string.
;12376          12461  2            ! Doubled up quotes are reduced to a single quote within the string
```

```
;12377      12462  2          ! and a carriage-return (end of line) is treated as an error (quotes
;12378      12463  2          ! not balanced).  Doubled up quotes set DOUBLED_QUOTES flag.
;12379      12464  2          !
;12380      12465  2          DOUBLED_QUOTES = FALSE;
;12381      12466  2          QUOTE = .CHARPTR[0];
;12382      12467  2          TOKENLEN = 0;
;12383      12468  2          TOKEN_TYPE[0] = TOKEN$K_STRING;
;12384      12469  2
;12385      12470  2          WHILE TRUE DO
;12386      12471  3              BEGIN
;12387      12472  3              CHARPTR = .CHARPTR + 1;
;12388      12473  3              IF .CHARPTR[0] EQL CAR_RET THEN SIGNAL(DBG$_MATQUOMIS);
;12389      12474  3              IF .CHARPTR[0] EQL .QUOTE
;12390      12475  3              THEN
;12391      12476  4                  BEGIN
;12392      12477  4                  CHARPTR = .CHARPTR + 1;
;12393      12478  4                  IF .CHARPTR[0] NEQ .QUOTE THEN EXITLOOP;
;12394      12479  4                  DOUBLED_QUOTES = TRUE;
;12395      12480  3                  END;
;12396      12481  3
;12397      12482  3              IF .TOKENLEN GEQ 255 THEN SIGNAL(DBG$_QUOSTRLONG);
;12398      12483  3              TOKENLEN = .TOKENLEN + 1;
;12399      12484  3              TOKENBUFFER[.TOKENLEN] = .CHARPTR[0];
;12400      12485  2              END;
;12401      12486  2
;12402      12487  2
;12403      12488  2          ! If langusge is PL/I, then check for a bit-string.
;12404      12489  2          ! If the quotes are "'" and there were no doubled up quotes, then
;12405      12490  2          ! see if the next character is the letter "B" or "b".  If so, then
;12406      12491  2          ! change TOKEN_TYPE to be a bit-string (TOKEN$K_BIT_STRING).  The
;12407      12492  2          ! "B" is not part of the length of the string.
;12408      12493  2          !
;12409      12494  2          IF .DBG$GB_LANGUAGE EQL DBG$K_PLI
;12410      12495  2              AND .QUOTE EQL DBG$K_QUOTE
;12411      12496  2              AND NOT .DOUBLED_QUOTES
;12412      12497  2          THEN
;12413      12498  2              IF .CHARPTR[0] EQL %C'B' OR .CHARPTR[0] EQL %C'b'
;12414      12499  2              THEN
;12415      12500  3                  BEGIN
;12416      12501  3                  CHARPTR = .CHARPTR + 1;
;12417      12502  3                  TOKEN_TYPE[0] = TOKEN$K_BIT_STRING;
;12418      12503  2                  END;
;12419      12504  2
;12420      12505  2
;12421      12506  2          ! We found the end of the string.  Complete the Counted ASCII string in
;12422      12507  2          ! the TOKENBUFFER buffer by filling in the length and return.
;12423      12508  2          !
;12424      12509  2          TOKENBUFFER[0] = .TOKENLEN;
;12425      12510  2          RETURN;
;12426      12511  2
;12427      12512  1          END;


                                        007C 00000 SCAN_QUOTED_STRING:
```

```
                                                                .WORD   Save R2,R3,R4,R5,R6                         ; 1240
                          56 00000000G  00  9E 00002            MOVAB   LIB$SIGNAL, R6                              ;
                          55 00000000'  EF  9E 00009            MOVAB   CHARPTR, R5                                 ;
                                        54  D4 00010            CLRL    DOUBLED_QUOTES                              ; 1246
                          53        00  B5  9A 00012            MOVZBL  @CHARPTR, QUOTE                             ; 1246
                                        52  D4 00016            CLRL    TOKENLEN                                    ; 1246
                     08   BC            02  D0 00018            MOVL    #2, @TOKEN_TYPE                             ; 1246
                                        65  D6 0001C 1$:        INCL    CHARPTR                                     ; 1247
                          0D        00  B5  91 0001E            CMPB    @CHARPTR, #13                               ; 1247
                                        09  12 00022            BNEQ    2$                                          ;
                             00028E30   8F  DD 00024            PUSHL   #167472                                     ;
                          66            01  FB 0002A            CALLS   #1, LIB$SIGNAL                              ;
      53        00   B5   08            00  ED 0002D 2$:        CMPZV   #0, #8, @CHARPTR, QUOTE                     ; 1247
                                        0D  12 00033            BNEQ    3$                                          ;
                                        65  D6 00035            INCL    CHARPTR                                     ; 1247
      53        00   B5   08            00  ED 00037            CMPZV   #0, #8, @CHARPTR, QUOTE                     ; 1247
                                        1F  12 0003D            BNEQ    5$                                          ;
                          54            01  D0 0003F            MOVL    #1, DOUBLED_QUOTES                          ; 1247
              000000FF    8F            52  D1 00042 3$:        CMPL    TOKENLEN, #255                              ; 1248
                                        09  19 00049            BLSS    4$                                          ;
                             000289DA   8F  DD 0004B            PUSHL   #166362                                     ;
                          66            01  FB 00051            CALLS   #1, LIB$SIGNAL                              ;
                                        52  D6 00054 4$:        INCL    TOKENLEN                                    ; 1248
                     04 BC42        00  B5  90 00056            MOVB    @CHARPTR, @TOKENBUFFER[TOKENLEN]            ; 1248
                                        BE  11 0005C            BRB     1$                                          ; 1247
                          05 00000000G  00  91 0005E 5$:        CMPB    DBG$GB_LANGUAGE, #5                         ; 1249
                                        1C  12 00065            BNEQ    7$                                          ;
                          27            53  D1 00067            CMPL    QUOTE, #39                                  ; 1249
                                        17  12 0006A            BNEQ    7$                                          ;
                          14            54  E8 0006C            BLBS    DOUBLED_QUOTES, 7$                          ; 1249
                     42   8F        00  B5  91 0006F            CMPB    @CHARPTR, #66                               ; 1249
                                        07  13 00074            BEQL    6$                                          ;
                     62   8F        00  B5  91 00076            CMPB    @CHARPTR, #98                               ;
                                        06  12 0007B            BNEQ    7$                                          ;
                                        65  D6 0007D 6$:        INCL    CHARPTR                                     ; 1250
                     08   BC            0D  D0 0007F            MOVL    #13, @TOKEN_TYPE                            ; 1250
                     04   BC            52  90 00083 7$:        MOVB    TOKENLEN, @TOKENBUFFER                      ; 1250
                                            04 00087            RET                                                 ; 1251

; Routine Size:  136 bytes,    Routine Base:  DBG$CODE + 4205


;12428          12513  1
;12429          12514  0       END ELUDOM



                                                                .EXTRN  LIB$SIGNAL

;                                 PSECT SUMMARY
;
;       Name                      Bytes                        Attributes
;
;   DBG$GLOBAL                        4  NOVEC,  WRT,  RD ,NOEXE,NOSHR,  LCL,  REL,  CON,  PIC,ALIGN(2)
;   DBG$OWN                        1344  NOVEC,  WRT,  RD ,NOEXE,NOSHR,  LCL,  REL,  CON,  PIC,ALIGN(2)
;   DBG$PLIT                      13389  NOVEC,NOWRT,  RD ,  EXE,  SHR,  LCL,  REL,  CON,  PIC,ALIGN(0)
```

```
;  DBG$CODE                    17037  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        36         0        1000        00:02.0
;  _$255$DUA28:[DEBUG.OBJ]STRUCDEF.L32;1     32         2         6           7        00:00.1
;  _$255$DUA28:[DEBUG.OBJ]DBGLIB.L32;1     1545       490        31          97        00:02.0
;  _$255$DUA28:[DEBUG.OBJ]DSTRECRDS.L32;1
;                                           418       125        29          31        00:00.4
;  _$255$DUA28:[DEBUG.OBJ]DBGMSG.L32;1      386        57        14          22        00:00.3
;  _$255$DUA28:[DEBUG.OBJ]DBGGEN.L32;1      150         1         0          12        00:00.3


; Information:  2
; Warnings:     0
; Errors:       0


;                                  COMMAND QUALIFIERS

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

; Size:         17037 code + 14737 data bytes
; Run Time:       07:23.7
; Elapsed Time:   17:24.9
; Lines/CPU Min:   1692
; Lexemes/CPU-Min: 28123
; Memory Used:  1461 pages
; Compilation Complete
```

DBGNTYPE
LIS

DBGNSHOW
LIS

DBGPARSER
LIS

0090 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

0091 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

DBGPERMOP
LIS