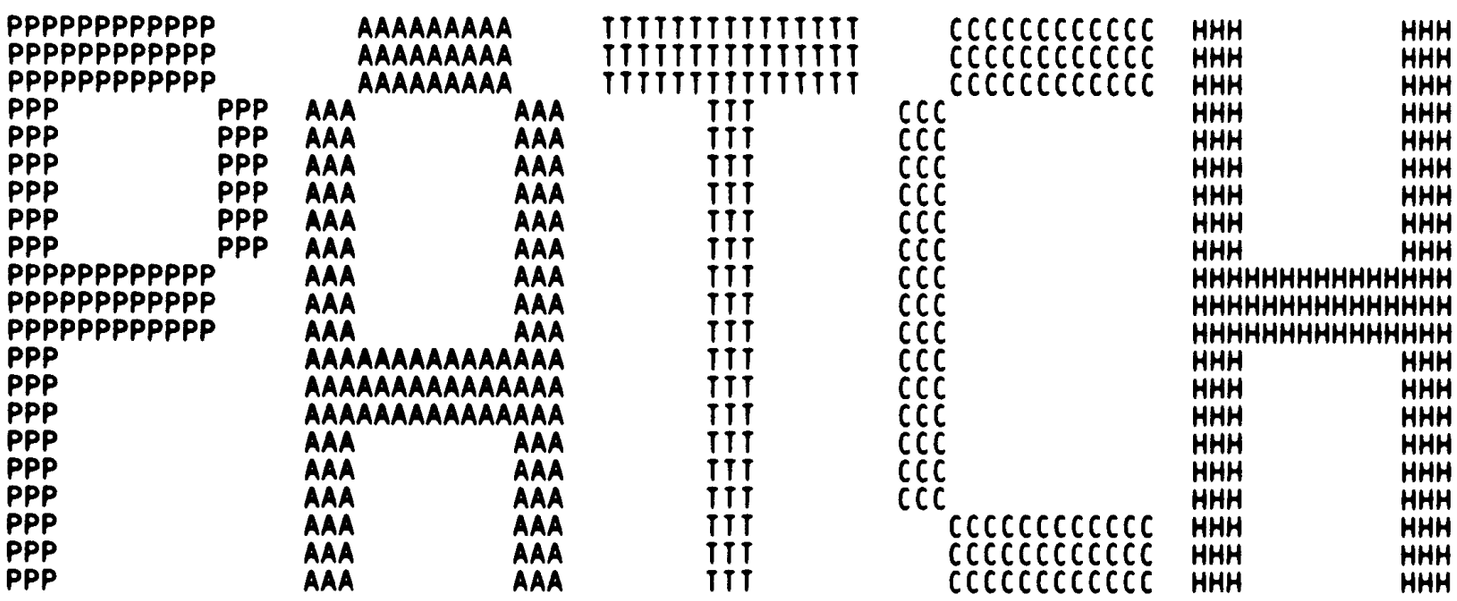
PPPPPPPPPPPP AAAAAAAAA TTTTTTTTTTTTTTT CCCCCCCCCCCC HHH HHH
PPP PPP AAA AAA TTT CCC HHH HHH
PPPPPPPPPPPP AAAAAAAAAAAAAAAA TTT CCC HHHHHHHHHHHHHHHH
PPP AAA AAA TTT CCC HHH HHH
PPP AAA AAA TTT CCCCCCCCCCCC HHH HHH

FILEID**PATRST

```
PPPPPPPP      AAAAAA    TTTTTTTTTT  RRRRRRRR      SSSSSSSS  TTTTTTTTTT
PPPPPPPP      AAAAAA    TTTTTTTTTT  RRRRRRRR      SSSSSSSS  TTTTTTTTTT
PP      PP   AA    AA       TT      RR      RR  SS            TT
PP      PP   AA    AA       TT      RR      RR  SS            TT
PP      PP   AA    AA       TT      RR      RR  SS            TT
PP      PP   AA    AA       TT      RR      RR  SS            TT
PPPPPPPP     AA    AA       TT      RRRRRRRR      SSSSSS      TT
PPPPPPPP     AA    AA       TT      RRRRRRRR      SSSSSS      TT
PP           AAAAAAAAAA     TT      RR  RR              SS    TT
PP           AAAAAAAAAA     TT      RR  RR              SS    TT
PP           AA    AA       TT      RR    RR            SS    TT          ....
PP           AA    AA       TT      RR    RR            SS    TT          ....
PP           AA    AA       TT      RR      RR  SSSSSSSS      TT          ....
PP           AA    AA       TT      RR      RR  SSSSSSSS      TT          ....

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 PATRST (
;     2        L 0002  0                 %IF %VARIANT EQL 1
;     3          0003  0                 %THEN
;     4          0004  0                         ADDRESSING_MODE (EXTERNAL = LONG_RELATIVE, NONEXTERNAL = LONG_RELATIVE),
;     5          0005  0                 %FI
;     6          0006  0                 IDENT = 'V04-000'
;     7          0007  0                 ) =
;     8          0008  1 BEGIN
;     9          0009  1
;    10          0010  1 !**************************************************************************
;    11          0011  1 !*                                                                        *
;    12          0012  1 !*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                               *
;    13          0013  1 !*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                 *
;    14          0014  1 !*  ALL RIGHTS RESERVED.                                                   *
;    15          0015  1 !*                                                                        *
;    16          0016  1 !*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
;    17          0017  1 !*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE  *
;    18          0018  1 !*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
;    19          0019  1 !*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
;    20          0020  1 !*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
;    21          0021  1 !*  TRANSFERRED.                                                           *
;    22          0022  1 !*                                                                        *
;    23          0023  1 !*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
;    24          0024  1 !*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
;    25          0025  1 !*  CORPORATION.                                                           *
;    26          0026  1 !*                                                                        *
;    27          0027  1 !*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
;    28          0028  1 !*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
;    29          0029  1 !*                                                                        *
;    30          0030  1 !*                                                                        *
;    31          0031  1 !**************************************************************************
;    32          0032  1
;    33          0033  1 !++
;    34          0034  1 ! FACILITY:     PATCH
;    35          0035  1 !
;    36          0036  1 ! ABSTRACT:     Use the Runtime Symbol Table (RST) data structures.
;    37          0037  1 !
;    38          0038  1 !
;    39          0039  1 ! ENVIRONMENT:  This module runs on VAX under VAX/VMS, user mode, non-AST level.
;    40          0040  1 !
;    41          0041  1 ! Author:       Kevin Pammett, August 18, 1977.
;    42          0042  1 !
;    43          0043  1 ! Version:  V02-014
;    44          0044  1 !
;    45          0045  1 ! MODIFIED BY:
;    46          0046  1 !
;    47          0047  1 !       V03-001 MTR0012         Mike Rhodes             16-Aug-1982                                    ; 1
;    48          0048  1 !               Modify file names to remove duplicate file name useage
;    49          0049  1 !               between code and require files.
;    50          0050  1 !
;    51          0051  1 !       V02-014 PCG0001         Peter George            02-FEB-1981
;    52          0052  1 !               Add require statement for LIB$:PATDEF.REQ
;    53          0053  1 !
;    54          0054  1 ! MODIFICATIONS:
;    55          0055  1 !
;    56          0056  1 !  NO   DATE            PROGRAMMER              PURPOSE
;    57          0057  1 !  --   ----            ----------              -------
```

```
;    58          0058  1 !
;    59          0059  1 !  00    21-DEC-77       K.D. MORSE      ADAPT VERSION 30 FOR PATCH.
;    60          0060  1 !  01    4-JAN-78        K.D. MORSE      MAKE PAT$DST_VALUE DISTINGUISH
;    61          0061  1 !                                        BETWEEN WHAT IT CAN'T EVALUATE
;    62          0062  1 !                                        BECAUSE OF ERROR AND BECAUSE THE
;    63          0063  1 !                                        GIVEN CONTEXT (REGISTER SET) IS
;    64          0064  1 !                                        INAPPROPRIATE. (31)
;    65          0065  1 !  02    24-JAN-78       K.D. MORSE      NO CHANGES FOR 32.
;    66          0066  1 !  03    02-MAR-78       K.D. MORSE      NO CHANGES FOR 33.
;    67          0067  1 !  04    24-MAR-78       K.D. MORSE      NO CHANGES FOR 34.
;    68          0068  1 !  05    28-MAR-78       K.D. MORSE      PAT$DST_VALUE NOW ACCEPTS
;    69          0069  1 !                                        DESCRIPTORS WHOSE  ACCESS
;    70          0070  1 !                                        FIELD = 2 FOR THE PC;  THESE
;    71          0071  1 !                                        ARE FIXED-POSITION DESCS. (35)
;    72          0072  1 !  06    30-MAR-78       K.D. MORSE      ADD RETURN STATUS CODE OF 3
;    73          0073  1 !                                        TO INDICATE ARRAY DESCRIPTORS
;    74          0074  1 !                                        FROM PAT$SYM_TO_VALU AND
;    75          0075  1 !                                        PAT$SYMBOL_VALU.  THIS IS SO
;    76          0076  1 !                                        THAT PAT$SYM_TO_VAL CAN
;    77          0077  1 !                                        ALWAYS RETURN THE ADDRESS BOUND
;    78          0078  1 !                                        TO A SYMBOL.
;    79          0079  1 !  07    06-APR-78       K.D. MORSE      LOOKUP_SYM NOW HANDLES SPECIAL
;    80          0080  1 !                                        EFFORT IN  NDING GLOBALS. (36)
;    81          0081  1 !                                        ROUTINES LOOKUP_GBL AND
;    82          0082  1 !                                        GBL_VAL_TO_SAT ADDED. (36)
;    83          0083  1 !                                        NEW MC_GBL FIELD 'LOCKED'.  IT
;    84          0084  1 !                                        MEANS THAT THE LOOKUP RTNS ARE
;    85          0085  1 !                                        NOT TO CONSULT THE GST. (37)
;    86          0086  1 !                                        NO MORE ACCS_DESCRIPTOR'S THAT
;    87          0087  1 !                                        POINT TO DESCRIPTORS IN THE USER'S
;    88          0088  1 !                                        IMAGE FROM FORTRAN. (38)
;    89          0089  1 !  08    25-APR-78       K.D. MORSE      CONVERT TO NATIVE COMPILER.
;    90          0090  1 !  09    17-MAY-78       K.D. MORSE      NO CHANGES FOR VERS 39-40.
;    91          0091  1 !  10    18-MAY-78       K.D. MORSE      NO CHANGES FOR VERS 41.
;    92          0092  1 !  11    13-JUN-78       K.D. MORSE      ADD FAO COUNTS TO SIGNALS.
;    93          0093  1 !  12    15-JUN-78       K.D. MORSE      PAT$SYM_TO_VALU RETURNS 0 OR 2
;    94          0094  1 !                                        FOR FAILURE. CHANGE CHECK IN
;    95          0095  1 !                                        PAT$SYM_TO_VAL TO TEST FOR LOW BIT.
;    96          0096  1 !  13    28-JUN-78       K.D. MORSE      NO CHANGES FOR VERS 42-43.
;    97          0097  1 !
;    98          0098  1 !--
```

```
;  100     0099  1 !
;  101     0100  1 ! TABLE OF CONTENTS:
;  102     0101  1 !
;  103     0102  1
;  104     0103  1 FORWARD ROUTINE
;  105     0104  1         PAT$ADD_NT_T_PV,                ! Build a pathanme vector by
;  106     0105  1                                         ! following an NT scope chain.
;  107     0106  1         PAT$LOOKUP_SYM,                 ! Find all occurrences of a given
;  108     0107  1                                         ! symbol in the RST data base.
;  109     0108  1         LOOKUP_GBL,                     ! Find all occurrences of a given
;  110     0109  1                                         ! global symbol in the GST.
;  111     0110  1         GBL_VAL_TO_SAT,                 ! Translate VALUE to a global symbol.
;  112     0111  1         PAT$GET_NXT_DUP,                ! Scan along hash chains.
;  113     0112  1         PAT$SYM_TO_VA',                 ! LOOK UP A FULLY QUALIFIED SYMBOL.
;  114     0113  1         PAT$SYM_TO_VALU,                ! Return more info than sym_to_val,
;  115     0114  1                                         ! but otherwise do the same thing.
;  116     0115  1         CONCAT_PATHS : NOVALUE,         ! Concatenate two pathname vectors together.
;  117     0116  1         PAT$NT_HASH_FCN,                ! Hashing function for calculating
;  118     0117  1                                         ! dispersal of NT entries.
;  119     0118  1         PATH_MATCH,                     ! Match a pathname to the implied
;  120     0119  1                                         ! path associated with an NT record.
;  121     0120  1         PAT$VAL_TO_SYM,                 ! Translate values to PATCH-time symbols.
;  122     0121  1         LOOKUP_LVT,                     ! Search the LVT.
;  123     0122  1         VAL_TO_SAT,                     ! Search the SAT.
;  124     0123  1         PAT$SYMBOL_VALU,                ! Associate symbols with corresponding value
;  125     0124  1         PAT$DST_VALUE,                  ! Deduce the value associated with a DST
;  126     0125  1         STD_SYM_EVAL;                   ! Evaluate dynamic symbols which use
;  127     0126  1                                         ! a standard encoding for how to do
;  128     0127  1                                         ! this evaluation.
;  129     0128  1
;  130     0129  1 !
;  131     0130  1 ! INCLUDE FILES:
;  132     0131  1 !
;  133     0132  1
;  134     0133  1 LIBPARY 'SYS$LIBRARY:LIB.L32';
;  135     0134  1 REQUIRE 'SRC$:PATPCT.REQ';
;  136     0174  1 REQUIRE 'SRC$:SYSSER.REQ';
```

```
;  R0206  1      SWITCHES LIST (SOURCE);
;  R0207  1
;  R0208  1      EXTERNAL ROUTINE
;  R0209  1              PAT$fao_out;            ! formats a line and outputs to the terminal
;  R0210  1
```

```
;  137          0256  1 REQUIRE 'SRC$:VXSMAC.REQ';
;  138          0321  1 REQUIRE 'SRC$:PATGEN.REQ';
;  139          0543  1 REQUIRE 'SRC$:PATRTS.REQ';
;  140          1639  1 REQUIRE 'LIB$:PATDEF.REQ';                    ! Defines literals
;  141          1693  1 REQUIRE 'LIB$:PATMSG.REQ';
;  142          1867  1
;  143          1868  1 !
;  144          1869  1 ! MACROS:
;  145          1870  1 !
;  146          1871  1
;  147          1872  1 !
;  148          1873  1 ! EQUATED SYMBOLS:
;  149          1874  1 !
;  150          1875  1
;  151          1876  1
;  152          1877  1 !
;  153          1878  1 ! OWN STORAGE:
;  154          1879  1 !
;  155          1880  1
;  156          1881  1 !
;  157          1882  1 ! EXTERNAL REFERENCES:
;  158          1883  1 !
;  159          1884  1
;  160          1885  1 EXTERNAL
;  161          1886  1         PAT$GL_MC_PTR    : REF MC_RECORD,          ! Pointer to the Module Chain (MC).
;  162          1887  1         PAT$GL_NT_HASH   : REF RST_POINTER,        ! Pointer to the name table (NT) hash vector
;  163          1888  1         PAT$GB_MOD_PTR : REF VECTOR[,BYTE],
;  164          1889  1         PAT$GL_CSP_PTR : REF PATHNAME_VECTOR;
;  165          1890  1
;  166          1891  1 EXTERNAL ROUTINE
;  167          1892  1         PAT$GET_NXT_SAT,                           ! Provide access to the SAT
;  168          1893  1         PAT$GET_NXT_LVT,                           ! Provide access to the LVT
;  169          1894  1         PAT$GET_NXT_GST,                           ! Scan the global symbol table.
;  170          1895  1         PAT$GET_DST_REC,                           ! Make a certain DST record available.
;  171          1896  1         PAT$MAP_ADDR;                              ! Map program address
;  172          1897  1
```

```
;  174          1898  1 GLOBAL ROUTINE PAT$ADD_NT_T_PV( NT_PTR, PV_PTR ) =
;  175          1899  1
;  176          1900  1 !++
;  177          1901  1 ! Functional Description:
;  178          1902  1 !
;  179          1903  1 !       Recur down thru an NT scope chain to build a pathname vector to
;  180          1904  1 !       correspond to the pathname implied by the scope chain.
;  181          1905  1 !
;  182          1906  1 ! Formal Parameters:
;  183          1907  1 !
;  184          1908  1 !  NT_PTR       -a pointer to the NT record where
;  185          1909  1 !                the symbol name is contained.  This is where the
;  186          1910  1 !                so-called 'scope chain' begins, if you consider such
;  187          1911  1 !                a pathname to go right to left.
;  188          1912  1 !  PV_PTR       -a pointer to somewhere in the pathname vector we are
;  189          1913  1 !                building.  Specifically, this points to where we should
;  190          1914  1 !                store a pointer to the first part (MODULE)
;  191          1915  1 !                of the pathname.
;  192          1916  1 !
;  193          1917  1 ! Implicit Inputs:
;  194          1918  1 !
;  195          1919  1 !       NT scope chains end either when the NT type is MODULE, or when the
;  196          1920  1 !       UP_SCOPE pointer is 0.  (The latter is what we do for NT records which
;  197          1921  1 !       come in to PATCH as global of DEFine symbols).                          ; (
;  198          1922  1 !
;  199          1923  1 !       Scope chains, as built by simply chaining thru NT records,
;  200          1924  1 !       are 'reverse pathnames', as defined by PATHNAME_VECTOR.  This
;  201          1925  1 !       is why this routine is recursive - we must recur down to
;  202          1926  1 !       the end of the NT chain befcre we can begin returning
;  203          1927  1 !       and filling in the pathname vector.
;  204          1928  1 !
;  205          1929  1 ! Implicit Outputs:
;  206          1930  1 !
;  207          1931  1 !       None.
;  208          1932  1 !
;  209          1933  1 ! Return Value:
;  210          1934  1 !
;  211          1935  1 !       The address (supposedly within the pathname vector)
;  212          1936  1 !       of where the next cs_pointer should be stored.
;  213          1937  1 !
;  214          1938  1 ! Side Effects:
;  215          1939  1 !
;  216          1940  1 !       This routine will blow up if the NT chain implies a
;  217          1941  1 !       pathname that is longer than is expected by the
;  218          1942  1 !       PATHNAME_VECTOR declaration.
;  219          1943  1 !
;  220          1944  1 !--
;  221          1945  2 BEGIN
;  222          1946  2
;  223          1947  2 MAP
;  224          1948  2         NT_PTR : REF NT_RECORD,
;  225          1949  2         PV_PTR : REF PATHNAME_VECTOR;
;  226          1950  2
;  227          1951  2 !++
;  228          1952  2 ! There's not much we can do till we've reached the end of the NT scope chain.
;  229          1953  2 !--
;  230          1954  3 IF (.NT_PTR[NT_TYPE] NEQ DSC$K_DTYPE_MOD) AND (.NT_PTR[NT_UP_SCOPE] NEQ 0)
```

```
;  231          1955  2 THEN
;  232          1956  2         PV_PTR = PAT$ADD_NT_T_PV( .NT_PTR[NT_UP_SCOPE], .PV_PTR );
;  233          1957  2
;  234          1958  2 !++
;  235          1959  2 ! The recursive call has modified our idea of where in the pathvector we
;  236          1960  2 ! should put the name pointer of the current pathname element.  Once MODULE has
;  237          1961  2 ! been reached, though, backing up (and out) is straightforward.
;  238          1962  2 !--
;  239          1963  2 PV_PTR[0] = NT_PTR[NT_NAME_CS];
;  240          1964  2
;  241          1965  2 !++
;  242          1966  2 ! The next element's CS pointer goes in the next higher pathname vector element.
;  243          1967  2 ! Since we do not recur on the last time around (i.e. after the last ROUTINE
;  244          1968  2 ! name was inserted), and since we must guarantee to end the pathname vector
;  245          1969  2 ! with a 0 cs pointer, we zero each potential 'next' entry here to ensure this.
;  246          1970  2 ! This is the point where damage will be done if/when someone gives a pathname
;  247          1971  2 ! that is too long, since the recursion does not 'count' how many times it fills
;  248          1972  2 ! in a PATHNAME_VECTOR entry.
;  249          1973  2 !--
;  250          1974  2 PV_PTR[1] = 0;
;  251          1975  2 RETURN( PV_PTR[1] );
;  252          1976  1 END;

                                                        .TITLE  PATRST
                                                        .IDENT  \V04-000\

                                                        .EXTRN  PAT$FAO_OUT, PAT$GL_MC_PTR
                                                        .EXTRN  PAT$GL_NT_HASH, PAT$GB_MOD_PTR
                                                        .EXTRN  PAT$GL_CSP_PTR, PAT$GET_NXT_SAT
                                                        .EXTRN  PAT$GET_NXT_LVT
                                                        .EXTRN  PAT$GET_NXT_GST
                                                        .EXTRN  PAT$GET_DST_REC
                                                        .EXTRN  PAT$MAP_ADDR, PAT$GL_RST_BEGN

                                                        .PSECT  _PAT$CODE,NOWRT,2

                                                 0004 00000     .ENTRY  PAT$ADD_NT_T_PV, Save R2                ; 1898
                    52 00000000G   EF   9E 00002        MOVAB   PAT$GL_RST_BEGN, R2
          50        04   AC        62   C1 00009        ADDL3   PAT$GL_RST_BEGN, NT_PTR, R0             ; 1954
                    BC   8F   02   A0   91 0000E        CMPB    2(R0), #188
                                   14   13 00013        BEQL    1$
                              08   A0   B5 00015        TSTW    8(R0)
                                   0F   13 00018        BEQL    1$
                              08   AC   DD 0001A        PUSHL   PV_PTR                                  ; 1956
                         7E   08   A0   3C 0001D        MOVZWL  8(R0), -(SP)
                    DB   AF        02   FB 00021        CALLS   #2, PAT$ADD_NT_T_PV
                    08   AC        50   D0 00025        MOVL    R0, PV_PTR
                         50   08   AC   D0 00029 1$:     MOVL    PV_PTR, R0                              ; 1963
          51        04   AC        62   C1 0002D        ADDL3   PAT$GL_RST_BEGN, NT_PTR, R1
                         80   0C   A1   9E 00032        MOVAB   12(R1), (R0)+
                                   60   D4 00036        CLRL    (R0)                                    ; 1974
                                        04 00038        RET                                             ; 1976

; Routine Size:  57 bytes,    Routine Base:  _PAT$CODE + 0000
```

```
;  254          1977  1 GLOBAL ROUTINE PAT$LOOKUP_SYM( SYM_CS ) =
;  255          1978  1
;  256          1979  1 !++
;  257          1980  1 ! Functional Description:
;  258          1981  1 !
;  259          1982  1 !       Look for a given symbol (NOT symbol pathname, only the symbol name
;  260          1983  1 !       itself) in the RST's name table (NT) always, and in the GST, sometimes.
;  261          1984  1 !       The latter happens when the MC_GBL_LOCKED field is FALSE.
;  262          1985  1 !
;  263          1986  1 !       Since there may be several such pathnames, this routine actually passes
;  264          1987  1 !       back a pointer to a so-called duplication chain.  This chain must be
;  265          1988  1 !       whatever GET_NXT_DUP requires to 'track down' successive duplicates.
;  266          1989  1 !
;  267          1990  1 ! Formal Parameters:
;  268          1991  1 !
;  269          1992  1 !       SYM_CS          -a counted string pointer to the symbol we are
;  270          1993  1 !                        to look for in the NT.
;  271          1994  1 !
;  272          1995  1 ! Implicit Inputs:
;  273          1996  1 !
;  274          1997  1 !       none.
;  275          1998  1 !
;  276          1999  1 ! Implicit Outputs:
;  277          2000  1 !
;  278          2001  1 !       none.
;  279          2002  1 !
;  280          2003  1 ! Return Value:
;  281          2004  1 !
;  282          2005  1 !
;  283          2006  1 !       FALSE, when there is no NT or GST entry for 'sym_cs',
;  284          2007  1 !       an NT-pointer to (a portion of) the hash chain which
;  285          2008  1 !       begins with an NT record which has 'sym_cs' as its symbol,
;  286          2009  1 !       otherwise.  If a global is found to match, its 'fake' NT record begins
;  287          2010  1 !       the hash chain.
;  288          2011  1 !
;  289          2012  1 ! Side Effects:
;  290          2013  1 !
;  291          2014  1 !       The GST may be searched linearly until the end or until SYM_CS is found.
;  292          2015  1 !--
;  293          2016  1
;  294          2017  2 BEGIN
;  295          2018  2
;  296          2019  2 MAP
;  297          2020  2         SYM_CS : CS_POINTER;                            ! A pointer to the symbol we are to look for
;  298          2021  2
;  299          2022  2 LOCAL
;  300          2023  2         GST_RECRD : REF GST_RECORD,                     ! Pointer to the global symbol table record
;  301          2024  2         NT_HASH,                                        ! Hash code for the given symbol.
;  302          2025  2         NT_PTR  : REF NT_RECORD;                        ! Pointer we go thru the NT with.
;  303          2026  2
;  304          2027  2 !++
;  305          2028  2 ! Access to the NT is via hashing.  If there is no pointer in the hash vector
;  306          2029  2 ! we get to, then no symbols exist that hash to the same value as the symbol
;  307          2030  2 ! we are looking up.  Otherwise we follow the hash chain and pass back a pointer
;  308          2031  2 ! to the first symbol we find which matches 'sym_cs'.
;  309          2032  2 !--
;  310          2033  2 NT_HASH = PAT$NT_HASH_FCN( .SYM_CS );
```

```
;  311     2034  2
;  312     2035  2 !++
;  313     2036  2 ! First see if a global symbol can be found by the given name in the GST.
;  314     2037  2 !--
;  315     2038  2 IF ((GST_RECRD = LOOKUP_GBL(.SYM_CS)) NEQ 0) AND
;  316     2039  3     ((NT_PTR = .PAT$GL_MC_PTR[ MC_GBL_NT_PTR ]) NEQ 0)
;  317     2040  2 THEN
;  318     2041  3         BEGIN
;  319     2042  3         LOCAL
;  320     2043  3                 global_type,
;  321     2044  3                 NAME_CS : CS_POINTER;
;  322     2045  3
;  323     2046  3         !++
;  324     2047  3         ! Found a global symbol.  Fill in the fake NT record for this global
;  325     2048  3         ! so that the rest of the sym_to_val routines don't have to worry about
;  326     2049  3         ! the special casing for globals.  The symbol name is in a different
;  327     2050  3         ! place depending on which type of GSD this is.  Pick up a pointer
;  328     2051  3         ! to this name, and record the GSD type.
;  329     2052  3         !--
;  330     2053  4         IF (.GST_RECRD[GST_ENTRY_TYPE] EQL GST_GLOBAL_DEFN)
;  331     2054  3         THEN
;  332     2055  4                 BEGIN
;  333     2056  4                 NAME_CS = GST_RECRD[GST_G_NAME_CS];
;  334     2057  4                 GLOBAL_TYPE = DSC$K_DTYPE_GBL;
;  335     2058  4                 END
;  336     2059  3         ELSE
;  337     2060  4                 BEGIN
;  338     2061  4                 NAME_CS = GST_RECRD[GST_E_NAME_CS];
;  339     2062  4                 GLOBAL_TYPE = DSC$K_DTYPE_ENT;
;  340     2063  3                 END;
;  341     2064  3
;  342     2065  3         !++
;  343     2066  3         ! Move the relevant fields into the NT record already reserved and
;  344     2067  3         ! pointer to by a field in the global MC record.
;  345     2068  3         !--
;  346     2069  3         CH$MOVE( .NAME_CS[0]+1, NAME_CS[0], NT_PTR[NT_NAME_CS]);
;  347     2070  3         NT_PTR[NT_TYPE] = .GLOBAL_TYPE;
;  348     2071  3         NT_PTR[NT_IS_GLOBAL] = TRUE;
;  349     2072  3         NT_PTR[NT_UP_SCOPE] = 0;
;  350     2073  3         NT_PTR[NT_GBL_VALUE] = .GST_RECRD[GST_VALUE];
;  351     2074  3
;  352     2075  3         !++
;  353     2076  3         ! Make this NT point forward to the hash chain in the RST for symbols
;  354     2077  3         ! which hash to this same value.  Whether or not there are any such
;  355     2078  3         ! symbols, we have still built a duplication chain.  We carefully don't
;  356     2079  3         ! change the existing hash chain so that nothing in the RST is actually
;  357     2080  3         ! changed by the addition of this new fake NT for the found global symbol.
;  358     2081  3         !--
;  359     2082  3         NT_PTR[NT_FORWARD] = .PAT$GL_NT_HASH[.NT_HASH];
;  360     2083  3
;  361     2084  3         !++
;  362     2085  3         ! The duplication chain has been built.
;  363     2086  3         !--
;  364     2087  3         RETURN(.NT_PTR);
;  365     2088  2         END;
;  366     2089  2
;  367     2090  2 !++
```

```
;  368          2091  2 ! Access to the NT is via hashing.  If there is no pointer in the hash vector we
;  369          2092  2 ! get to, then no symbols exist that hash to the same value as the symbol we are
;  370          2093  2 ! looking up.  Otherwise we follow the hash chain and pass back a pointer to the
;  371          2094  2 ! first symbol we find which matches 'sym_cs'.
;  372          2095  2 !--
;  373          2096  3 IF ((NT_PTR = .PAT$GL_NT_HASH[.NT_HASH]) EQL 0)
;  374          2097  2 THEN
;  375          2098  2         !++
;  376          2099  2         ! If there is no hash chain, then there is no 'sym_cs' in the
;  377          2100  2         ! RST data base.
;  378          2101  2         !--
;  379          2102  2         RETURN(FALSE);
;  380          2103  2
;  381          2104  2 !++
;  382          2105  2 ! There does exist a hash chain for the given symbol. If a match does exist,
;  383          2106  2 ! it must be in this chain, so we simply follow along it and compare the symbol
;  384          2107  2 ! names found therein for the first match.
;  385          2108  2 !--
;  386          2109  2 DO
;  387          2110  3         BEGIN
;  388          2111  3         !++
;  389          2112  3         ! We are only interested in this entry if the symbol names actually do match.
;  390          2113  3         !--
;  391          2114  4         IF (CH$EQL(.SYM_CS[0],SYM_CS[1],.NT_PTR[NT_NAME_CS],NT_PTR[NT_NAME_ADDR]))
;  392          2115  3         THEN
;  393          2116  4                 BEGIN
;  394          2117  4                 !++
;  395          2118  4                 ! This is the place in the chain that we want to pass back a pointer to.
;  396          2119  4                 !--
;  397          2120  4                 RETURN(.NT_PTR);
;  398          2121  3                 END;
;  399          2122  3
;  400          2123  3         !++
;  401          2124  3         ! Otherwise just skip along the chain. The hash chain ends when the
;  402          2125  3         ! NT_FORWARD pointer is 0.
;  403          2126  3         !--
;  404          2127  3         END
;  405          2128  2 WHILE( (NT_PTR = .NT_PTR[NT_FORWARD]) NEQ 0 );
;  406          2129  2
;  407          2130  2 !++
;  408          2131  2 ! If we fall out of the above loop, we ran off the end of the hash chain
;  409          2132  2 ! without finding a match to the symbol we were looking for.
;  410          2133  2 !--
;  411          2134  2 RETURN(FALSE);
;  412          2135  1 END;

                                    0FFC 00000          .ENTR   PAT$LOOKUP_SYM, Save R2,R3,R4,R5,R6,R7,R8,- ; 1977
                                                                R9,R10,R11
                    5B 00000000G    EF   9E 00002       MOVAB   PAT$GL_RST_BEGN, R11                        ;
                               04   AC   DD 00009       PUSHL   SYM_CS                                      ; 2033
          00000000V EF             01   FB 0000C       CALLS   #1, PAT$NI_HASH_FCN                         ;
                    59             50   D0 00013       MOVL    R0, NT_HASH                                 ;
                               04   AC   DD 00016       PUSHL   SYM_CS                                      ; 2038
```

```
                    00000000V  EF              01  FB 00019          CALLS   #1, LOOKUP_GBL                                  ;
                               58              50  D0 00020          MOVL    R0, GST_RECRD                                  ;
                                               4D  13 00023          BEQL    3$                                             ;
                    50 00000000G  EF            6B  C1 00025          ADDL3   PAT$GL_RST_BEGN, PAT$GL_MC_PTR, R0     ; 2039
                               56       04      A0  3C 0002D          MOVZWL  4(R0), NT_PTR                          ;
                                               3F  13 00031          BEQL    3$                                     ;
                                   01          68  91 00033          CMPB    (GST_RECRD), #1                        ; 2053
                                               0A  12 00036          BNEQ    1$                                     ;
                                   50      09  A8  9E 00038          MOVAB   9(R8), NAME_CS                         ; 2056
                                   5A      B4  8F  9A 0003C          MOVZBL  #180, GLOBAL_TYPE                      ; 2057
                                               08  11 00040          BRB     2$                                     ; 2053
                                   50      0B  A8  9E 00042  1$:     MOVAB   11(R8), NAME_CS                        ; 2061
                                   5A      B3  8F  9A 00046          MOVZBL  #179, GLOBAL_TYPE                      ; 2062
                                   51          60  9A 0004A  2$:     MOVZBL  (NAME_CS), R1                          ; 2069
                                               51  D6 0004D          INCL    R1                                     ;
                               57              56  C1 0004F          ADDL3   PAT$GL_RST_BEGN, NT_PTR, R7            ;
                        0C     A7              60  28 00053          MOVC3   R1, (NAME_CS), 12(R7)                  ;
                                   02      A7  5A  90 00058          MOVB    GLOBAL_TYPE, 2(R7)                     ; 2070
                                   03      A7  01  88 0005C          BISB2   #1, 3(R7)                              ; 2071
                                           08  A7  B4 00060          CLRW    8(R7)                                  ; 2072
                                   04      A7  05  A8 00063          MOVL    5(GST_RECRD), 4(R7)                    ; 2073
                                   67 00000000GFF  49  B0 00068          MOVW    @PAT$GL_NT_HASH[NT_HASH], (R7)         ; 2082
                                               24  11 00070          BRB     5$                                     ; 2087
                                   56 00000000GFF  49  3C 00072  3$:     MOVZWL  @PAT$GL_NT_HASH[NT_HASH], NT_PTR       ; 2096
                                               23  13 0007A          BEQL    7$                                     ;
                                   54      04  AC  D0 0007C          MOVL    SYM_CS, R4                             ; 2114
                                   51      04  BC  9A 00080  4$:     MOVZBL  @SYM_CS, R1                            ;
                               57              56  C1 00084          ADDL3   PAT$GL_RST_BEGN, NT_PTR, R7            ;
                                   50      0C  A7  9A 00088          MOVZBL  12(R7), R0                             ;
                    50             00      01  A4  2D 0008C          CMPC5   R1, 1(R4), #0, R0, 13(R7)              ;
                                           0D  A7      00092                                                         ;
                                               04  12 00094          BNEQ    6$                                     ;
                                   50              56  D0 00096  5$:     MOVL    NT_PTR, R0                             ; 2120
                                                   04 00099          RET                                            ;
                                   56              67  3C 0009A  6$:     MOVZWL  (R7), NT_PTR                           ; 2128
                                                   E1  12 0009D          BNEQ    4$                                     ;
                                                   50  D4 0009F  7$:     CLRL    R0                                     ; 2135
                                                       04 000A1          RET                                            ;

; Routine Size:  162 bytes,    Routine Base:  _PAT$CODE + 0039
```

```
;  414          2136  1 ROUTINE LOOKUP_GBL( SYM_CS ) =
;  415          2137  1
;  416          2138  1 !++
;  417          2139  1 ! Functional Description:
;  418          2140  1 !
;  419          2141  1 !       Look for a given symbol (NOT symbol pathname, only the
;  420          2142  1 !       symbol name itself) in the Global symbol table (GST).
;  421          2143  1 !
;  422          2144  1 ! Formal Parameters:
;  423          2145  1 !
;  424          2146  1 !       SYM_CS          -a counted string pointer to the symbol we are
;  425          2147  1 !                        to look for in the GST.
;  426          2148  1 !
;  427          2149  1 ! Implicit Inputs:
;  428          2150  1 !
;  429          2151  1 !
;  430          2152  1 !       We don't consult the GST if MC_GBL_LOCKED is TRUE.
;  431          2153  1 !
;  432          2154  1 !       GET_NXT_GST is all set up to allow us to
;  433          2155  1 !       read thru the (mapped) GST sequentially.
;  434          2156  1 !
;  435          2157  1 ! Implicit Outputs:
;  436          2158  1 !       none.
;  437          2159  1 !
;  438          2160  1 ! Return Value:
;  439          2161  1 !
;  440          2162  1 !       0, when no GST entry for 'sym_cs' can be found,
;  441          2163  1 !       a pointer to the GST record for
;  442          2164  1 !       the found symbol, otherwise.
;  443          2165  1 !
;  444          2166  1 ! Side Effects:
;  445          2167  1 !
;  446          2168  1 !       The GST may be searched sequentially until either
;  447          2169  1 !       the end is encounered, or the symbol is found.
;  448          2170  1 !--
;  449          2171  1
;  450          2172  2 BEGIN
;  451          2173  2
;  452          2174  2 MAP
;  453          2175  2         SYM_CS : CS_POINTER;
;  454          2176  2
;  455          2177  2 LOCAL
;  456          2178  2         GST_RECRD : REF GST_RECORD;                             ! Pointer to where a fetched GST record live
;  457          2179  2
;  458          2180  2 !++
;  459          2181  2 ! Don't even look in the GST if it is locked.  This happens either because the
;  460          2182  2 ! caller specifically does not want a global, or because the caller is using the
;  461          2183  2 ! 'cache' NT and SAT records for globals, and doesn't want them overwritten.
;  462          2184  2 !--
;  463          2185  3 IF (.PAT$GL_MC_PTR[MC_GBL_LOCKED])
;  464          2186  2 THEN
;  465          2187  2         RETURN(0);
;  466          2188  2
;  467          2189  2 !++
;  468          2190  2 ! Process the GST records sequentially, giving up if some error occurs.
;  469          2191  2 ! First, do an INIT so that subsequent GETs return record pointers starting
;  470          2192  2 ! from the beginning.
```

```
;  471     2193  2 !--
;  472     2194  2 PAT$GET_NXT_GST(1);
;  473     2195  2
;  474     2196  2 !++
;  475     2197  2 ! PAT$GET_NXT_GST returns 0 when there are no more GST records to process.
;  476     2198  2 ! Otherwise it returns a pointer to each successive GST record.
;  477     2199  2 !--
;  478     2200  3 WHILE ((GST_RECRD = PAT$GET_NXT_GST(0)) NEQ 0)
;  479     2201  2 DO
;  480     2202  3         BEGIN
;  481     2203  3         !++
;  482     2204  3         ! We process each record depending on its GST type.
;  483     2205  3         !--
;  484     2206  3         CASE .GST_RECRD[GST_ENTRY_TYPE] FROM GST_LOWEST TO GST_HIGHEST OF
;  485     2207  3             SET
;  486     2208  3
;  487     2209  3             [ GST_GLOBAL_DEFN ]:                                ! Definition of a global symbol.
;  488     2210  4                 BEGIN
;  489     2211  4                 !++
;  490     2212  4                 ! Check for string match of given and indicated symbols.
;  491     2213  4                 !--
;  492     2214  5                 IF (CH$EQL(.SYM_CS[0], SYM_CS[1], .GST_RECRD[GST_G_NAME_CS],
;  493     2215  5                            GST_RECRD[GST_G_NAME_ADDR]))
;  494     2216  4                 THEN
;  495     2217  4                         !++
;  496     2218  4                         ! Found the right record, and we know there are no more
;  497     2219  4                         ! (because these are globals).
;  498     2220  4                         !--
;  499     2221  4                         EXITLOOP;
;  500     2222  3                 END;
;  501     2223  3
;  502     2224  3             [ GST_ENTRY_DEFN ]:                                 ! Definition of a entry point.
;  503     2225  4                 BEGIN
;  504     2226  4                 !++
;  505     2227  4                 ! Check for string match of given and indicated symbols.
;  506     2228  4                 !--
;  507     2229  5                 IF (CH$EQL(.SYM_CS[0], SYM_CS[1], .GST_RECRD[GST_E_NAME_CS],
;  508     2230  5                            GST_RECRD[GST_E_NAME_ADDR]))
;  509     2231  4                 THEN
;  510     2232  4                         !++
;  511     2233  4                         ! Found the right record, and we know there are no more
;  512     2234  4                         ! (because these are globals).
;  513     2235  4                         !--
;  514     2236  4                         EXITLOOP;
;  515     2237  3                 END;
;  516     2238  3
;  517     2239  3             [INRANGE, OUTRANGE]:                                ! Error.
;  518     2240  4                 BEGIN
;  519     2241  4                 RETURN(0);
;  520     2242  3                 END;
;  521     2243  3
;  522     2244  3             TES;
;  523     2245  3
;  524     2246  3         !++
;  525     2247  3         ! Go back and process the next record.
;  526     2248  3         !--
;  527     2249  2         END;
```

```
;  528          2250  2
;  529          2251  2 !++
;  530          2252  2 ! If the above WHILE exits, then either we encountered the normal end of GST
;  531          2253  2 ! processing, or we found the symbol match.  Pass back the address of the given
;  532          2254  2 ! symbol's record, of 0, the failure code.
;  533          2255  2 !--
;  534          2256  2 RETURN(.GST_RECRD);
;  535          2257  1 END;


                                                003C 00000 LOOKUP_GBL:
                                                           .WORD   Save R2,R3,R4,R5                                ; 2136
                          55 00000000G   EF   9E 00002     MOVAB   PAT$GET_NXT_GST, R5
               50 00000000G   EF 00000000G   EF   C1 00009 ADDL3   PAT$GL_RST_BEGN, PAT$GL_MC_PTR, R0            ; 2185
               49        03   A0             02   E0 00015 BBS     #2, 3(R0), 7$
                                               01   DD 0001A PUSHL   #1                                          ; 2194
                          65                   01   FB 0001C CALLS   #1, PAT$GET_NXT_GST
                                               7E   D4 0001F 1$:    CLRL    -(SP)                                  ; 2200
                          65                   01   FB 00021 CALLS   #1, PAT$GET_NXT_GST
                          54                   50   D0 00024 MOVL    R0, GST_RECRD
                                               36   13 00027 BEQL    6$
                  02               01          64   8F 00029 CASEB   (GST_RECRD), #1, #2                         ; 2206
                0036             001D        0008       0002D 2$:    .WORD   3$-2$,-
                                                                             4$-2$,-
                                                                             7$-2$
                                               2E   11 00033 BRB     7$                                          ; 2241
                          50        04         AC   D0 00035 3$:    MOVL    SYM_CS, R0                          ; 2214
                          52                   60   9A 00039 MOVZBL  (R0), R2
                          51        09         A4   9A 0003C MOVZBL  9(GST_RECRD), R1
          51              00        01   A0        52   2D 00040 CMPC5   R2, 1(R0), #0, R1, 10(GST_RECRD)       ; 2215
                                         0A   A4        00046
                                               13   11 00048 BRB     5$
                          50        04         AC   D0 0004A 4$:    MOVL    SYM_CS, R0                          ; 2229
                          52                   60   9A 0004E MOVZBL  (R0), R2
                          51        0B         A4   9A 00051 MOVZBL  11(GST_RECRD), R1
          51              00        01   A0        52   2D 00055 CMPC5   R2, 1(R0), #0, R1, 12(GST_RECRD)       ; 2230
                                         0C   A4        0005B
                                               C0   12 0005D 5$:    BNEQ    1$
                          50                   54   D0 0005F 6$:    MOVL    GST_RECRD, R0                       ; 2256
                                                    04 00062        RET
                                               50   D4 00063 7$     CLRL    R0                                  ; 2257
                                                    04 00065        RET

; Routine Size:  102 bytes,    Routine Base:  _PAT$CODE + 00DB
```

```
;  537          2258  1 ROUTINE GBL_VAL_TO_SAT( VALUE ) =
;  538          2259  1
;  539          2260  1 !++
;  540          2261  1 ! Functional Description:
;  541          2262  1 !
;  542          2263  1 !       Search the GST for the closest global symbol match
;  543          2264  1 !       to the given value.  If found, build a temporary SAT
;  544          2265  1 !       and NT entry for this symbol so that the rest
;  545          2266  1 !       of the RST manipulating routine do not have to
;  546          2267  1 !       special-case globals.
;  547          2268  1 !
;  548          2269  1 ! Formal Parameters:
;  549          2270  1 !
;  550          2271  1 !       VALUE   -The key value to match to Global symbol.
;  551          2272  1 !
;  552          2273  1 ! Implicit Inputs:
;  553          2274  1 !
;  554          2275  1 !       We don't consult the GST if MC_GBL_LOCKED.
;  555          2276  1 !       This happens because we normally want
;  556          2277  1 !       to let VAL_TO_SYM consult globals, but to we need
;  557          2278  1 !       to override this when PC_RULE calls it.
;  558          2279  1 !
;  559          2280  1 !       GET_NXT_GST is all set up to all us to
;  560          2281  1 !       read thru the (mapped) GST sequentially.
;  561          2282  1 !
;  562          2283  1 !       The first MC record is reserved for manipulating
;  563          2284  1 !       globals.  As such, it has two fields which permanently
;  564          2285  1 !       point to the 'fake' NT and SAT records which get
;  565          2286  1 !       filled by this routine (and others).
;  566          2287  1 !
;  567          2288  1 ! Implicit Outputs:
;  568          2289  1 !
;  569          2290  1 !       If a match is found, the MC_GBL NT and SAT records
;  570          2291  1 !       are filled in so that they are usable just like
;  571          2292  1 !       other NT/SAT pairs are.
;  572          2293  1 !
;  573          2294  1 ! Return Value:
;  574          2295  1 !
;  575          2296  1 !       FALSE, when no candidate is found,
;  576          2297  1 !       a pointer to the SAT record for
;  577          2298  1 !       the found symbol, otherwise.
;  578          2299  1 !
;  579          2300  1 ! Side Effects:
;  580          2301  1 !
;  581          2302  1 !       The GST may be searched sequentially until either
;  582          2303  1 !       the end is encountered, or an exact match is found.
;  583          2304  1 !--
;  584          2305  1
;  585          2306  2 BEGIN
;  586          2307  2
;  587          2308  2 LOCAL
;  588          2309  2         GLOBAL_TYPE,
;  589          2310  2         NAME_CS : CS_POINTER,
;  590          2311  2         GST_RECRD : REF GST_RECORD,                    ! Pointer to where a fetched GST record live
;  591          2312  2         GBL_SAT_PTR : REF SAT_RECORD,                  ! Pointers to the SAT and NT records we buil
;  592          2313  2         GBL_NT_PTR : REF NT_RECORD,
;  593          2314  2         BEST_MATCH : REF GST_RECORD;                   ! Pointer to so-far 'best' match found.
```

```
;  594     2315  2
;  595     2316  2 !++
;  596     2317  2 ! If we find a match candidate, we will use the GLOBAL temporary SAT and NT
;  597     2318  2 ! records to pass on the needed info.  If no space has been reserved for these
;  598     2319  2 ! records, we must give up now.  Moreover, if this space is currently in use
;  599     2320  2 ! (for SYM_TO_VAL) then again we must give up.
;  600     2321  2 !--
;  601     2322  3 IF (.PAT$GL_MC_PTR[MC_GBL_LOCKED])
;  602     2323  2 THEN
;  603     2324  2         RETURN(0);
;  604     2325  3 IF ((GBL_SAT_PTR = .PAT$GL_MC_PTR[MC_GBL_SAT_PTR]) EQL 0)
;  605     2326  2 THEN
;  606     2327  2         RETURN(0);
;  607     2328  3 IF ((GBL_NT_PTR = .PAT$GL_MC_PTR[MC_GBL_NT_PTR]) EQL 0)
;  608     2329  2 THEN
;  609     2330  2         RETURN(0);
;  610     2331  2
;  611     2332  2 !++
;  612     2333  2 ! Process the GST records sequentially, giving up if some error occurs.
;  613     2334  2 ! First, do an INIT so that subsequent GETs return record pointers starting from
;  614     2335  2 ! the beginning.
;  615     2336  2 !--
;  616     2337  2 PAT$GET_NXT_GST(1);
;  617     2338  2 BEST_MATCH = 0;
;  618     2339  2
;  619     2340  2 !++
;  620     2341  2 ! PAT$GET_NXT_GST returns 0 when there are no more GST records to process.
;  621     2342  2 !--
;  622     2343  3 WHILE ((GST_RECRD = PAT$GET_NXT_GST(0)) NEQ 0)
;  623     2344  2 DO
;  624     2345  3         BEGIN
;  625     2346  3         !++
;  626     2347  3         ! We process each record depending on its GST type.
;  627     2348  3         !--
;  628     2349  3         CASE .GST_RECRD[GST_ENTRY_TYPE] FROM GST_LOWEST TO GST_HIGHEST OF
;  629     2350  3             SET
;  630     2351  3
;  631     2352  3             [ GST_GLOBAL_DEFN ,                                    ! Definition of a global symbol.
;  632     2353  3               GST_ENTRY_DEFN ]:                                    ! Definition of global ENTRY points.
;  633     2354  4                 BEGIN
;  634     2355  4                 !++
;  635     2356  4                 ! Note that the only two types we support can be handled
;  636     2357  4                 ! together because the GST_VALUE field is in the same place in
;  637     2358  4                 ! both records.  First check for an exact match because then
;  638     2359  4                 ! we can abandon any further looking.
;  639     2360  4                 !--
;  640     2361  5                 IF (.VALUE EQLA .GST_RECRD[GST_VALUE])
;  641     2362  4                 THEN
;  642     2363  5                         BEGIN
;  643     2364  5                         BEST_MATCH = .GST_RECRD;
;  644     2365  5                         EXITLOOP;
;  645     2366  4                         END;
;  646     2367  4
;  647     2368  4                 !++
;  648     2369  4                 ! Inexact matches are still better than nothing.
;  649     2370  4                 !--
;  650     2371  5                 IF (.VALUE GTRA .GST_RECRD[GST_VALUE])
```

```
;  651     2372  4                 THEN
;  652     2373  4                     BEGIN
;  653     2374  4                     !++
;  654     2375  4                     ! A match.  See if we already have one.
;  655     2376  5                     !--
;  656     2377  4                     IF (.BEST_MATCH EQL 0)
;  657     2378  4                     THEN
;  658     2379  4                         !++
;  659     2380  4                         ! Any one is better than none.
;  660     2381  4                         !--
;  661     2382  4                         BEST_MATCH = .GST_RECRD
;  662     2383  4                     ELSE
;  663     2384  4                         !++
;  664     2385  4                         ! Take the new one only if this symbol
;  665     2386  4                         ! is closer than the previous best one.
;  666     2387  5                         !--
;  667     2388  4                         IF (.BEST_MATCH[GST_VALUE] LSSA .GST_RECRD[GST_VALUE])
;  668     2389  4                         THEN
;  669     2390  3                             BEST_MATCH = .GST_RECRD;
;  670     2391  3                     END;
;  671     2392  3
;  672     2393  4                 [INRANGE, OUTRANGE]:                    ! Error.
;  673     2394  4                     BEGIN
;  674     2395  3                     RETURN(FALSE);
;  675     2396  3                     END;
;  676     2397  3
;  677     2398  3                 TES;
;  678     2399  3
;  679     2400  3             !++
;  680     2401  3             ! Go back and process the next record.
;  681     2402  2             !--
;  682     2403  2             END;
;  683     2404  2
;  684     2405  2         !++
;  685     2406  2         ! If the above WHILE exits, then we encountered the normal end of GST processing.
;  686     2407  2         ! If we didn't find any possible match, return failure status.
;  687     2408  3         !--
;  688     2409  2         IF (.BEST_MATCH EQL 0)
;  689     2410  2         THEN
;  690     2411  2             RETURN(0);
;  691     2412  2
;  692     2413  2         !++
;  693     2414  2         ! Success - a candidate has been found.  Fill in the required SAT and NT
;  694     2415  2         ! records, and pass back a pointer to the former.  These records have already
;  695     2416  2         ! been (permanently) allocated space - they are pointed to by fields in the
;  696     2417  2         ! global MC record.
;  697     2418  2         !--
;  698     2419  2         GBL_SAT_PTR[SAT_LB] = .BEST_MATCH[GST_VALUE];
;  699     2420  2         GBL_SAT_PTR[SAT_UB] = 0;
;  700     2421  2         GBL_SAT_PTR[SAT_NT_PTR] = .GBL_NT_PTR;
;  701     2422  2
;  702     2423  2         !++
;  703     2424  2         ! The symbol name is in a different place depending on which type of GSD this
;  704     2425  2         ! is.  Pick up a pointer to this name, and record the GSD type.
;  705     2426  3         !--
;  706     2427  2         IF (.BEST_MATCH[GST_ENTRY_TYPE] EQL GST_GLOBAL_DEFN)
;  707     2428  3         THEN
;                               BEGIN
```

```
;  708          2429  3          NAME_CS = BEST_MATCH[GST_G_NAME_CS];
;  709          2430  3          GLOBAL_TYPE = DSC$K_DTYPE_GBL;
;  710          2431  3          END
;  711          2432  2 ELSE
;  712          2433  3          BEGIN
;  713          2434  3          NAME_CS = BEST_MATCH[GST_E_NAME_CS];
;  714          2435  3          GLOBAL_TYPE = DSC$K_DTYPE_ENT;
;  715          2436  2          END;
;  716          2437  2 CH$MOVE( .NAME_CS[0]+1, NAME_CS[0], GBL_NT_PTR[NT_NAME_CS]);
;  717          2438  2 GBL_NT_PTR[NT_TYPE] = .GLOBAL_TYPE;
;  718          2439  2 GBL_NT_PTR[NT_IS_GLOBAL] = TRUE;
;  719          2440  2 GBL_NT_PTR[NT_GBL_VALUE] = .BEST_MATCH[GST_VALUE];
;  720          2441  2 GBL_NT_PTR[NT_FORWARD] = 0;
;  721          2442  2 GBL_NT_PTR[NT_UP_SCOPE] = 0;
;  722          2443  2 RETURN(.GBL_SAT_PTR);
;  723          2444  1 END;

                                 0FFC 00000 GBL_VAL_TO_SAT:
                                            .WORD   Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11          ; 2258
                    5B 00000000G   EF   9E 00002    MOVAB   PAT$GL_RST_BEGN, R11                 ;
                    5A 00000000G   EF   9E 00009    MOVAB   PAT$GET_NXT_GST, R10                 ;
          50 00000000G   EF              6B  C1 00010    ADDL3   PAT$GL_RST_BEGN, PAT$GL_MC_PTR, R0    ; 2322
          26        03   A0              02  E0 00018    BBS     #2, 3(R0), 3$                        ;
                    58        35   A0   D0  00001D     MOVL    53(R0), GBL_SAT_PTR                  ; 2325
                                   42   13  00021     BEQL    8$                                   ;
                    52        04   A0   3C  00023     MOVZWL  4(R0), GBL_NT_PTR                    ; 2328
                                   3C   13  00027     BEQL    8$                                   ;
                                   01   DD  00029     PUSHL   #1                                   ; 2337
                              6A        01   FB  0002B     CALLS   #1, PAT$GET_NXT_GST              ;
                                   56   D4  0002E     CLRL    BEST_MATCH                           ; 2338
                                   7E   D4  00030 1$:  CLRL    -(SP)                                ; 2343
                              6A        01   FB  00032     CALLS   #1, PAT$GET_NXT_GST              ;
                                   50   D5  00035     TSTL    GST_RECRD                            ;
                                   2A   13  00037     BEQL    7$                                   ;
          02             01             60  8F  00039     CASEB   (GST_RECRD), #1, #2                  ; 2349
        0070           0008           0008      0003D 2$:  .WORD   4$-2$,-                          ;
                                                                  4$-2$,-                           ;
                                                                  11$-2$                            ;
                                   68   11  00043 3$:  BRB     11$                                  ; 2394
                    05   A0        04   AC  01  00045 4$:  CMPL    VALUE, 5(GST_RECRD)              ; 2361
                                   05   12  0004A     BNEQ    5$                                   ;
                         56             50  D0  0004C     MOVL    GST_RECRD, BEST_MATCH            ; 2364
                                   12   11  0004F     BRB     7$                                   ; 2363
                                   DD   1B  00051 5$:  BLEQU   1$                                   ; 2371
                                   56   D5  00053     TSTL    BEST_MATCH                           ; 2376
                                   07   13  00055     BEQL    6$                                   ;
                    05   A0        05   A6  D1  00057     CMPL    5(BEST_MATCH), 5(GST_RECRD)      ; 2387
                                   D2   1E  0005C     BGEQU   1$                                   ;
                         56             50  D0  0005E 6$:  MOVL    GST_RECRD, BEST_MATCH            ; 2389
                                   CD   11  00061     BRB     1$                                   ; 2376
                                   56   D5  00063 7$:  TSTL    BEST_MATCH                           ; 2408
                                   46   13  00065 8$:  BEQL    11$                                  ;
                    02   A8        05   A6  D0  00067     MOVL    5(BEST_MATCH), 2(GBL_SAT_PTR)    ; 2418
```

```
                                06    A8  D4 0006C          CLRL    6(GBL_SAT_PTR)                          ; 2419
                            68      52  B0 0006F          MOVW    GBL_NT_PTR, (GBL_SAT_PTR)               ; 2420
                            01      66  91 00072          CMPB    (BEST_MATCH), #1                        ; 2426
                                    0A  12 00075          BNEQ    9$                                      ;
                            50  09  A6  9E 00077          MOVAB   9(R6), NAME_CS                          ; 2429
                            59  B4  8F  9A 0007B          MOVZBL  #180, GLOBAL_TYPE                       ; 2430
                                    08  11 0007F          BRB     10$                                     ; 2426
                            50  0B  A6  9E 00081 9$:      MOVAB   11(R6), NAME_CS                         ; 2434
                            59  B3  8F  9A 00085          MOVZBL  #179, GLOBAL_TYPE                       ; 2435
                            51      60  9A 00089 10$:     MOVZBL  (NAME_CS), R1                           ; 2437
                                    51  D6 0008C          INCL    R1                                      ;
                        57          52  6B  C1 0008E      ADDL3   PAT$GL_RST_BEGN, GBL_NT_PTR, R7         ;
                    0C  A7          60  51  28 00092      MOVC3   R1, (NAME_CS), 12(R7)                   ;
                            02  A7      59  90 00097      MOVB    GLOBAL_TYPE, 2(R7)                      ; 2438
                            03  A7      01  88 0009B      BISB2   #1, 3(R7)                               ; 2439
                            04  A7  05  A6  D0 0009F      MOVL    5(BEST_MATCH), 4(R7)                    ; 2440
                                    67  B4  000A4         CLRW    (R7)                                    ; 2441
                                08  A7  B4  000A6         CLRW    8(R7)                                   ; 2442
                            50      58  D0 000A9          MOVL    GBL_SAT_PTR, R0                         ; 2443
                                    04  000AC             RET                                             ;
                                    50  D4 000AD 11$:     CLRL    R0                                      ; 2444
                                        04 000AF          RET                                             ;

; Routine Size:  176 bytes,    Routine Base:  _PAT$CODE + 0141
```

```
;  725          2445  1 GLOBAL ROUTINE PAT$SYM_TO_VAL( PATH_VEC_PTR, VALUE_PTR ) =
;  726          2446  1
;  727          2447  1 !++
;  728          2448  1 ! Functional Description:
;  729          2449  1 !
;  730          2450  1 !       Use the RST/GST data base to translate a given symbol pathname to its
;  731          2451  1 !       corresponding value.
;  732          2452  1 !
;  733          2453  1 !       This routine is quite similar PAT$SYM_TO_VALU, except that it is called
;  734          2454  1 !       when ONLY the value is required.  As this builds in less knowledge of
;  735          2455  1 !       the RST and its data structures, this routine should be called whenever
;  736          2456  1 !       possible.
;  737          2457  1 !
;  738          2458  1 ! Formal Parameters:
;  739          2459  1 !
;  740          2460  1 !  PATH_VEC_PTR -a pointer to the pathname vector from a PATCH command
;  741          2461  1 !
;  742          2462  1 !  VALUE_PTR    -The location to hold the returned address bound to the
;  743          2463  1 !                symbol pathname.
;  744          2464  1 !
;  745          2465  1 ! Implicit Inputs:
;  746          2466  1 !
;  747          2467  1 !       The value to be passed back is a longword.                                  ;
;  748          2468  1 !
;  749          2469  1 ! Implicit Outputs:
;  750          2470  1 !
;  751          2471  1 !       None.
;  752          2472  1 !
;  753          2473  1 ! Return Value:
;  754          2474  1 !
;  755          2475  1 !       TRUE, if symbol was located successfully.
;  756          2476  1 !       FALSE, otherwise.
;  757          2477  1 !
;  758          2478  1 ! Side Effects:
;  759          2479  1 !
;  760          2480  1 !       The bound address is written into the return location.
;  761          2481  1 !
;  762          2482  1 !--
;  763          2483  1
;  764          2484  2 BEGIN
;  765          2485  2
;  766          2486  2 MAP
;  767          2487  2         PATH_VEC_PTR : REF PATHNAME_VECTOR,
;  768          2488  2         VALUE_PTR : REF VECTOR[,LONG];
;  769          2489  2
;  770          2490  2 LOCAL
;  771          2491  2         STATUS,                                         ! INDICATOR FOR NO SYMBOL (0), ADDR OF SYMBO
;  772          2492  2         DSC_PTR : REF BLOCK[,BYTE],                     ! Array descriptor address
;  773          2493  2         VALU_DESC : VALU_DESCRIPTOR;                    ! Local value descriptor
;  774          2494  2
;  775          2495  2 !++
;  776          2496  2 ! PAT$SYM_TO_VALU does all the work of searching the symbol table.  It returns
;  777          2497  2 ! either the address bound to the symbol (STATUS = 1), the address of a
;  778          2498  2 ! descriptor for an array (STATUS = 3), or an indication that the symbol was not
;  779          2499  2 ! found (STATUS = 0 OR 2).
;  780          2500  2 !--
;  781          2501  3 IF (STATUS =  PAT$SYM_TO_VALU(.PATH_VEC_PTR, VALU_DESC))
```

```
;  782          2502  2 THEN
;  783          2503  3         BEGIN
;  784          2504  3         !++
;  785          2505  3         ! Now determine if the value returned is the address bound to
;  786          2506  3         ! the symbol or the address of an array descriptor.
;  787          2507  3         !--
;  788          2508  4         IF (.STATUS EQL 1)
;  789          2509  3         THEN
;  790          2510  3                 VALUE_PTR[0] = .VALU_DESC[VALU_VALUE]
;  791          2511  3         ELSE
;  792          2512  4                 BEGIN
;  793          2513  4                 !++
;  794          2514  4                 ! This must be an array descriptor.  Return the address bound
;  795          2515  4                 ! to the symbol not the address of the descriptor.
;  796          2516  4                 !--
;  797          2517  4                 DSC_PTR = .VALU_DESC[VALU_VALUE];
;  798          2518  4                 VALUE_PTR[0] = .DSC_PTR[DSC$A_POINTER];
;  799          2519  3                 END;
;  800          2520  3         RETURN(TRUE);
;  801          2521  3         END
;  802          2522  2 ELSE
;  803          2523  2         !++
;  804          2524  2         ! Failure. There is really nothing further we can do.
;  805          2525  2         !--
;  806          2526  2         RETURN(FALSE);
;  807          2527  1 END;


                                    0000 00000          .ENTRY  PAT$SYM_TO_VAL, Save nothing                    ; 2445
                            5E      08  C2 00002          SUBL2   #8, SP
                                    5E  DD 00005          PUSHL   SP                                              ; 2501
                                04  AC  DD 00007          PUSHL   PATH_VEC_PTR
                    00000000V   EF      02  FB 0000A      CALLS   #2, PAT$SYM_TO_VALU
                            19          50  E9 00011      BLBC    STATUS, 3$
                            01          50  D1 00014      CMPL    STATUS, #1                                      ; 2508
                                        07  12 00017      BNEQ    1$
                        08  BC      02  AE  D0 00019      MOVL    VALU_DESC+2, @VALUE_PTR                         ; 2510
                                        09  11 0001E      BRB     2$
                            50      02  AE  D0 00020 1$:  MOVL    VALU_DESC+2, DSC_PTR                            ; 2517
                        08  BC      04  A0  D0 00024      MOVL    4(DSC_PTR), @VALUE_PTR                          ; 2518
                            50          01  D0 00029 2$:  MOVL    #1, R0                                          ; 2526
                                            04 0002C      RET
                                        50  D4 0002D 3$:  CLRL    R0
                                            04 0002F      RET                                                     ; 2527

; Routine Size:  48 bytes,    Routine Base:  _PAT$CODE + 01F1
```

```
;  809          2528  1 GLOBAL ROUTINE PAT$SYM_TO_VALU( PATH_VEC_PTR, VALUE_DESC_ADDR ) =
;  810          2529  1
;  811          2530  1 !++
;  812          2531  1 ! Functional Description:
;  813          2532  1 !
;  814          2533  1 !       Use the RST/GST data base to translate a given
;  815          2534  1 !       symbol pathname to its corresponding value.
;  816          2535  1 !       Build and return a so-called 'value descriptor'
;  817          2536  1 !       which corresponds to this found value.  It is
;  818          2537  1 !       within this routine that the notion of "search rules"
;  819          2538  1 !       is implemented.
;  820          2539  1 !
;  821          2540  1 ! Formal Parameters:
;  822          2541  1 !
;  823          2542  1 !  PATH_VEC_PTR -a pointer to the pathname vector which the user
;  824          2543  1 !                input as the symbol name.  This can be a simple
;  825          2544  1 !                "name", a compound "rout\rout1 ...\name",
;  826          2545  1 !                or a fully-qualified "module\...\name".
;  827          2546  1 !
;  828          2547  1 !  VALUE_DESC_ADDR      -The address of a value descriptor which
;  829          2548  1 !                         we are to 'fill in' with the one which
;  830          2549  1 !                         corresponds to the value we find associated
;  831          2550  1 !                         with the given symbol.
;  832          2551  1 !
;  833          2552  1 ! Implicit Inputs:
;  834          2553  1 !
;  835          2554  1 !       The value to be passed back is a longword, or at least,
;  836          2555  1 !       the current definition of VALU_DESCRIPTOR facilitates
;  837          2556  1 !       passing around the value we find.
;  838          2557  1 !
;  839          2558  1 !       The current state of the MODE data structure
;  840          2559  1 !       is used, along with the built-in relationship between the
;  841          2560  1 !       setting of these bits and the implied search rules, to
;  842          2561  1 !       pick out which is the correct match from the RST.
;  843          2562  1 !
;  844          2563  1 !
;  845          2564  1 ! Implicit Outputs:
;  846          2565  1 !
;  847          2566  1 !       Whatever is implied by the current definition of VALU_DESCRIPTOR.
;  848          2567  1 !
;  849          2568  1 ! Return Value:
;  850          2569  1 !
;  851          2570  1 !       TRUE, if the symbol is found successfully and
;  852          2571  1 !                the returned value is the address bound to the symbol,
;  853          2572  1 !       3, if the symbol is found successfully and
;  854          2573  1 !                the returned value is the address of descriptor for an array,
;  855          2574  1 !       FALSE or 2, otherwise.
;  856          2575  1 !
;  857          2576  1 ! Side Effects:
;  858          2577  1 !
;  859          2578  1 !       none.
;  860          2579  1 !--
;  861          2580  1
;  862          2581  2 BEGIN
;  863          2582  2
;  864          2583  2 MAP
;  865          2584  2         PATH_VEC_PTR : REF PATHNAME_VECTOR,
```

```
;  866          2585  2          VALUE_DESC_ADDR : REF VALU_DESCRIPTOR;
;  867          2586  2
;  868          2587  2 LOCAL
;  869          2588  2          TEMP_PATH_VEC : PATHNAME_VECTOR,                    ! We use a local pathname vector to concat-
;  870          2589  2                                                             ! enate various other partial pathname
;  871          2590  2                                                             ! vectors into.
;  872          2591  2          CHAIN_PTR : REF NT_RECORD,                          ! A pointer to the hash chain which contains
;  873          2592  2                                                             ! all occurrences of the symbol which ends
;  874          2593  2                                                             ! the pathname pointed to by PATH_VEC_PTR.
;  875          2594  2          NT_PTR : REF NT_RECORD;                             ! We extract each RST-pointer from the
;  876          2595  2                                                             ! hash chain and use the local NT_PTR
;  877          2596  2                                                             ! to contain the entry.
;  878          2597  2
;  879          2598  2 LOCAL
;  880          2599  2          INDEX;                                             ! Index of symbol in path vector.
;  881          2600  2
;  882          2601  2 !++
;  883          2602  2 ! See that the path vector has at least one entry, and that the RST has been initialized.
;  884          2603  2 !--
;  885          2604  2 IF .path_vec_ptr EQL 0
;  886          2605  2 THEN
;  887          2606  2          RETURN FALSE;
;  888          2607  3 IF (.PATH_VEC_PTR[0] EQL 0) OR (.PAT$GL_MC_PTR EQL 0)
;  889          2608  2 THEN
;  890          2609  2          RETURN(FALSE);
;  891          2610  2
;  892          2611  2 !++
;  893          2612  2 ! The symbol is the last entry in the given pathname.
;  894          2613  2 ! Pick up what index this is into the pathname vector.
;  895          2614  2 !--
;  896          2615  2 INDEX = 0;
;  897          2616  2
;  898          2617  2 REPEAT
;  899          2618  3          BEGIN
;  900          2619  4          IF (.PATH_VEC_PTR[.INDEX+1] EQL 0)
;  901          2620  3          THEN
;  902          2621  3                    EXITLOOP;                                ! INDEX points to the symbol name.
;  903          2622  4          IF ((INDEX = .INDEX +1) GEQ MAX_PATH_SIZE)
;  904          2623  3          THEN
;  905          2624  3                    RETURN(FALSE);
;  906          2625  2          END;
;  907          2626  2
;  908          2627  2 !++
;  909          2628  2 ! Discover all occurrences of the given symbol in the RST data base.  If some
;  910          2629  2 ! error occurs, we must give up.  This is because we consider not finding any
;  911          2630  2 ! symbols an error.
;  912          2631  2 !--
;  913          2632  3 IF ((CHAIN_PTR = PAT$LOOKUP_SYM( .PATH_VEC_PTR[.INDEX] )) EQL 0)
;  914          2633  2 THEN
;  915          2634  2          RETURN(FALSE);                                     ! We can find no such symbol.
;  916          2635  2
;  917          2636  2 !++
;  918          2637  2 ! First, before we bother with all the search rules, see if the symbol happens
;  919          2638  2 ! to be unique.  To do this, setup to scan the duplication chain.  Grab the
;  920          2639  2 ! first one, and  accept it as long as there is no second one, and as long as
;  921          2640  2 ! we were given only a symbol name.
;  922          2641  2 !--
```

```
;  923     2642  3 IF (.INDEX EQL 0)
;  924     2643  2 THEN
;  925     2644  3         BEGIN
;  926     2645  3         !++
;  927     2646  3         ! We have only a symbol name.  Check for uniqueness.
;  928     2647  3         !--
;  929     2648  3         PAT$GET_NXT_DUP( 1, .CHAIN_PTR );
;  930     2649  3
;  931     2650  4         IF ((NT_PTR = PAT$GET_NXT_DUP(0)) EQL 0)
;  932     2651  3         THEN
;  933     2652  3                 RETURN(FALSE);                                  ! There is not even one.
;  934     2653  3
;  935     2654  4         IF (PAT$GET_NXT_DUP(0) EQL 0)
;  936     2655  3         THEN
;  937     2656  4                 BEGIN
;  938     2657  4                 !++
;  939     2658  4                 ! The symbol is unique.  Return a corresponding
;  940     2659  4                 ! value descriptor and status code.
;  941     2660  4                 !--
;  942     2661  4                 VALUE_DESC_ADDR[VALU_NT_PTR] = .NT_PTR;
;  943     2662  4                 RETURN( PAT$SYMBOL_VALUE( .NT_PTR, VALUE_DESC_ADDR[VALU_VALUE] ));
;  944     2663  3                 END;
;  945     2664  2         END;
;  946     2665  2
;  947     2666  2 !++
;  948     2667  2 ! We have all the duplicates, and there are  more than 1 of them.  Now see
;  949     2668  2 ! if one of them matches the given pathname.  Note that we have to apply each
;  950     2669  2 ! search rule, in order, to ALL of the duplicates before we can go on to the next
;  951     2670  2 ! search rule. (As opposed to applying all search rules to each consecutive
;  952     2671  2 ! secutive duplicate - which sounds tempting but doesn't work).
;  953     2672  2 !
;  954     2673  2 ! If GLOBAL or DEFine symbols are acceptable first, we can just try to match
;  955     2674  2 ! up the two pathnames  directly.
;  956     2675  2 !--
;  957     2676  3 IF (.PAT$GB_MOD_PTR[MODE_GLOBALS])
;  958     2677  2 THEN
;  959     2678  3         BEGIN
;  960     2679  3         !++
;  961     2680  3         ! Set up to scan the hash chain, and loop along
;  962     2681  3         ! it until we have considered all duplicates.
;  963     2682  3         !--
;  964     2683  3         PAT$GET_NXT_DUP( 1, .CHAIN_PTR);
;  965     2684  4         WHILE( (NT_PTR = PAT$GET_NXT_DUP(0)) NEQ 0 )
;  966     2685  3                 DO
;  967     2686  4                 BEGIN
;  968     2687  5                 IF (PATH_MATCH( .PATH_VEC_PTR, .NT_PTR))
;  969     2688  4                 THEN
;  970     2689  5                         BEGIN
;  971     2690  5                         !++
;  972     2691  5                         ! Return a value descriptor and status code.
;  973     2692  5                         !--
;  974     2693  5                         VALUE_DESC_ADDR[VALU_NT_PTR] = .NT_PTR;
;  975     2694  5                         RETURN( PAT$SYMBOL_VALUE( .NT_PTR, VALUE_DESC_ADDR[VALU_VALUE] ));
;  976     2695  4                         END;
;  977     2696  4
;  978     2697  3                 END;                                            ! Loop back and try the next duplicate.
;  979     2698  2         END;                                                    ! It is not a GLOBAL.
```

```
;  980          2699  2 !++
;  981          2700  2 ! Next, unless asked not to, we try qualifying the given pathname with the one
;  982          2701  2 ! that corresponds to  the user-set current scope position (CSP) vector.
;  983          2702  2 ! Note that we can't make this check if the CSP is null because this is
;  984          2703  2 ! equivalent to allowing a GLOBAL to satisfy the match which we can't allow
;  985          2704  2 ! at this point.
;  986          2705  2 !--
;  987          2706  3 IF (.PAT$GB_MOD_PTR[MODE_SCOPE] AND .PAT$GL_CSP_PTR NEQ 0)
;  988          2707  2 THEN
;  989          2708  3         BEGIN
;  990          2709  3         !++
;  991          2710  3         ! Set up to scan the hash chain, and loop along
;  992          2711  3         ! it until we have considered all duplicates.
;  993          2712  3         !--
;  994          2713  3         PAT$GET_NXT_DUP( 1, .CHAIN_PTR);
;  995          2714  4         WHILE( (NT_PTR = PAT$GET_NXT_DUP(0)) NEQ 0 )
;  996          2715  3                 DO
;  997          2716  4                 BEGIN
;  998          2717  4                 !++
;  999          2718  4                 ! First, build the new pathname by perpending the CSP.
;  1000         2719  4                 !--
;  1001         2720  4                 CONCAT_PATHS( TEMP_PATH_VEC, .PAT$GL_CSP_PTR, .PATH_VEC_PTR );
;  1002         2721  5                 IF (PATH_MATCH( TEMP_PATH_VEC, .NT_PTR))
;  1003         2722  4                 THEN
;  1004         2723  5                         BEGIN
;  1005         2724  5                         !++
;  1006         2725  5                         ! Return a value descriptor and status code.
;  1007         2726  5                         !--
;  1008         2727  5                         VALUE_DESC_ADDR[VALU_NT_PTR] = .NT_PTR;
;  1009         2728  5                         RETURN( PAT$SYMBOL_VALU( .NT_PTR, VALUE_DESC_ADDR[VALU_VALUE] ));
;  1010         2729  4                         END;
;  1011         2730  4
;  1012         2731  3                 END;                                    ! Go back and try the next duplicate.
;  1013         2732  2         END;                                            ! You can't find it via CSP.
;  1014         2733  2
;  1015         2734  2 !++
;  1016         2735  2 ! Next, if we haven't already considered globals, now's the time.
;  1017         2736  2 !--
;  1018         2737  3 IF (NOT .PAT$GB_MOD_PTR[MODE_GLOBALS])
;  1019         2738  2 THEN
;  1020         2739  3         BEGIN
;  1021         2740  3         !++
;  1022         2741  3         ! Set up to scan the hash chain, and loop along it until we have
;  1023         2742  3         ! considered all duplicates.
;  1024         2743  3         !--
;  1025         2744  3         PAT$GET_NXT_DUP( 1, .CHAIN_PTR);
;  1026         2745  4         WHILE( (NT_PTR = PAT$GET_NXT_DUP(0)) NEQ 0 )
;  1027         2746  3                 DO
;  1028         2747  4                 BEGIN
;  1029         2748  5                 IF (PATH_MATCH( .PATH_VEC_PTR, .NT_PTR))
;  1030         2749  4                 THEN
;  1031         2750  5                         BEGIN
;  1032         2751  5                         !++
;  1033         2752  5                         ! Return a value descriptor and status code.
;  1034         2753  5                         !--
;  1035         2754  5                         VALUE_DESC_ADDR[VALU_NT_PTR] = .NT_PTR;
;  1036         2755  5                         RETURN( PAT$SYMBOL_VALU( .NT_PTR, VALUE_DESC_ADDR[VALU_VALUE] ));
```

```
;  1037          2756  4                        END;
;  1038          2757  3                    END;                                ! Go back and try the next duplicate.
;  1039          2758  2            END;                                        ! It is not a GLOBAL.
;  1040          2759  2 !++
;  1041          2760  2 ! We applied all the search rules to all the duplicates and none was the one
;  1042          2761  2 ! we were looking for.
;  1043          2762  2 !--
;  1044          2763  2 RETURN(FALSE);
;  1045          2764  1 END;


                                01FC 00000          .ENTRY  PAT$SYM_TO_VALU, Save R2,R3,R4,R5,R6,R7,R8 ; 2528
                58 00000000G   EF   9E 00002        MOVAB   PAT$GL_CSP_PTR, R8                         ;
                57 00000000V   EF   9E 00009        MOVAB   PATH_MATCH, R7                             ;
                56 00000000G   EF   9E 00010        MOVAB   PAT$GB_MOD_PTR, R6                         ;
                55 00000000V   EF   9E 00017        MOVAB   PAT$GET_NXT_DUP, R5                        ;
                5E             2C   C2 0001E        SUBL2   #44, SP                                    ;
                53        04   AC   D0 00021        MOVL    PATH_VEC_PTR, R3                           ; 2604
                               1B   13 00025        BEQL    2$                                         ;
                               63   D5 00027        TSTL    (R3)                                       ; 2607
                               17   13 00029        BEQL    2$                                         ;
                   00000000G   EF   D5 0002B        TSTL    PAT$GL_MC_PTR                              ;
                               0F   13 00031        BEQL    2$                                         ;
                               52   D4 00033        CLRL    INDEX                                      ; 2615
                         04 A342   D5 00035 1$:     TSTL    4(R3)[INDEX]                               ; 2619
                               0A   13 00039        BEQL    3$                                         ;
                               52   D6 0003B        INCL    INDEX                                      ; 2622
                0A             52   D1 0003D        CMPL    INDEX, #10                                 ;
                               F3   19 00040        BLSS    1$                                         ;
                             00BB   31 00042 2$:    BRW     11$                                        ; 2624
                             6342   DD 00045 3$:    PUSHL   (R3)[INDEX]                                ; 2632
      FDCB      CF               01   FB 00048        CALLS   #1, PAT$LOOKUP_SYM                         ;
                54               50   D0 0004D        MOVL    R0, CHAIN_PTR                              ;
                                 F0   13 00050        BEQL    2$                                         ;
                                 52   D5 00052        TSTL    INDEX                                      ; 2642
                                 1A   12 00054        BNEQ    4$                                         ;
                                 54   DD 00056        PUSHL   CHAIN_PTR                                  ; 2648
                                 01   DD 00058        PUSHL   #1                                         ;
                               65     02   FB 0005A        CALLS   #2, PAT$GET_NXT_DUP                    ;
                                      7E   D4 0005D        CLRL    -(SP)                                  ; 2650
                               65     01   FB 0005F        CALLS   #1, PAT$GET_NXT_DUP                    ;
                               52     50   D0 00062        MOVL    R0, NT_PTR                             ;
                                      DB   13 00065        BEQL    2$                                     ;
                                      7E   D4 00067        CLRL    -(SP)                                  ; 2654
                               65     01   FB 00069        CALLS   #1, PAT$GET_NXT_DUP                    ;
                                      50   D5 0006C        TSTL    R0                                     ;
                                      7D   13 0006E        BEQL    10$                                    ;
                               50     66   D0 00070 4$:    MOVL    PAT$GB_MOD_PTR, R0                     ; 2676
                               1D  06 A0   E9 00073        BLBC    6(R0), 6$                              ;
                                      54   DD 00077        PUSHL   CHAIN_PTR                              ; 2683
                                      01   DD 00079        PUSHL   #1                                     ;
                               65     02   FB 0007B        CALLS   #2, PAT$GET_NXT_DUP                    ;
                                      7E   D4 0007E 5$:    CLRL    -(SP)                                  ; 2684
                               65     01   FB 00080        CALLS   #1, PAT$GET_NXT_DUP                    ;
```

```
                        52           50  D0 00083          MOVL    R0, NT_PTR                                      ;
                                     0C  13 00086          BEQL    6$                                              ;
                                     52  DD 00088          PUSHL   NT_PTR                                          ; 2687
                                     53  DD 0008A          PUSHL   R3                                              ;
                                 67      02  FB 0008C      CALLS   #2, PATH_MATCH                                  ;
                                 EC      50  E9 0008F      BLBC    R0, 5$                                          ;
                                         59  11 00092      BRB     10$                                             ; 2693
                                 50      66  D0 00094  6$: MOVL    PAT$GB_MOD_PTR, R0                              ; 2706
                                 30  05  A0  E9 00097      BLBC    5(R0), 8$                                       ;
                                         68  D5 0009B      TSTL    PAT$GL_CSP_PTR                                  ;
                                         2C  13 0009D      BEQL    8$                                              ;
                                         54  DD 0009F      PUSHL   CHAIN_PTR                                       ; 2713
                                         01  DD 000A1      PUSHL   #1                                              ;
                                 65      02  FB 000A3      CALLS   #2, PAT$GET_NXT_DUP                             ;
                                         7E  D4 000A6  7$: CLRL    -(SP)                                           ; 2714
                                 65      01  FB 000A8      CALLS   #1, PAT$GET_NXT_DUP                             ;
                                 52      50  D0 000AB      MOVL    R0, NT_PTR                                      ;
                                         1B  13 000AE      BEQL    8$                                              ;
                                         53  DD 000B0      PUSHL   R3                                              ; 2720
                                         68  DD 000B2      PUSHL   PAT$GL_CSP_PTR                                  ;
                                 08      AE  9F 000B4      PUSHAB  TEMP_PATH_VEC                                   ;
                00000000V        EF      03  FB 000B7      CALLS   #3, CONCAT_PATHS                                ;
                                         52  DD 000BE      PUSHL   NT_PTR                                          ; 2721
                                 04      AE  9F 000C0      PUSHAB  TEMP_PATH_VEC                                   ;
                                 67      02  FB 000C3      CALLS   #2, PATH_MATCH                                  ;
                                 DD      50  E9 000C6      BLBC    R0, 7$                                          ;
                                         22  11 000C9      BRB     10$                                             ; 2727
                                 50      66  D0 000CB  8$: MOVL    PAT$GB_MOD_PTR, R0                              ; 2737
                                 2E  06  A0  E8 000CE      BLBS    6(R0), 11$                                      ;
                                         54  DD 000D2      PUSHL   CHAIN_PTR                                       ; 2744
                                         01  DD 000D4      PUSHL   #1                                              ;
                                 65      02  FB 000D6      CALLS   #2, PAT$GET_NXT_DUP                             ;
                                         7E  D4 000D9  9$: CLRL    -(SP)                                           ; 2745
                                 65      01  FB 000DB      CALLS   #1, PAT$GET_NXT_DUP                             ;
                                 52      50  D0 000DE      MOVL    R0, NT_PTR                                      ;
                                         1D  13 000E1      BEQL    11$                                             ;
                                         52  DD 000E3      PUSHL   NT_PTR                                          ; 2748
                                         53  DD 000E5      PUSHL   R3                                              ;
                                 67      02  FB 000E7      CALLS   #2, PATH_MATCH                                  ;
                                 EC      50  E9 000EA      BLBC    R0, 9$                                          ;
                             08  BC      52  B0 000ED 10$: MOVW    NT_PTR, @VALUE_DESC_ADDR                        ; 2754
                         7E  08  AC      02  C1 000F1      ADDL3   #2, VALUE_DESC_ADDR, -(SP)                      ; 2755
                                         52  DD 000F6      PUSHL   NT_PTR                                          ;
                    00000000V    EF      02  FB 000F8      CALLS   #2, PAT$SYMBOL_VALU                             ;
                                             04  000FF      RET                                                     ;
                                         50  D4 00100 11$: CLRL    R0                                              ; 2764
                                             04  00102      RET                                                     ;

; Routine Size:  259 bytes,    Routine Base:  _PAT$CODE + 0221
```

```
;1047     2765  1 GLOBAL ROUTINE PAT$GET_NXT_DUP( INIT_FLAG, NEW_CHAIN ) =
;1048     2766  1
;1049     2767  1 !++
;1050     2768  1 ! Functional Description:
;1051     2769  1 !
;1052     2770  1 !       Facilitate going along hash chains (or whatever
;1053     2771  1 !       it is that symbol duplicates are stored in) when we only
;1054     2772  1 !       want to look at symbols which match a given symbol.
;1055     2773  1 !
;1056     2774  1 ! Formal Parameters:
;1057     2775  1 !
;1058     2776  1 !       INIT_FLAG       0 => we are in the middle of a chain so
;1059     2777  1 !                        pass back the next NT and ignore NEW_CHAIN.
;1060     2778  1 !                        (so that we don't have to always pass 2 parameters)
;1061     2779  1 !                       1 => this is a real init.  NEW_CHAIN marks the
;1062     2780  1 !                        beginning of the new chain we want to look at.
;1063     2781  1 !                       -The returned value is undefined for INIT_FLAG==1
;1064     2782  1 !                        type calls.  NEW_CHAIN is not defined, otherwise.
;1065     2783  1 !       NEW_CHAIN       -a pointer to a chain of forward-linked
;1066     2784  1 !                        NT records which all hash to the same value.
;1067     2785  1 !                        When an init is done, (INIT_FLAG == 1), the
;1068     2786  1 !                        first element on the new chain
;1069     2787  1 !                        is assumed to be the one that we henceforth
;1070     2788  1 !                        want to 'key' off.
;1071     2789  1 !
;1072     2790  1 ! Implicit Inputs:
;1073     2791  1 !
;1074     2792  1 !       -The way that one 'chains' along hash chains.
;1075     2793  1 !
;1076     2794  1 !       -The first element in the chain when INIT==1 is the
;1077     2795  1 !        'key' element for that chain.  The symbol name therein
;1078     2796  1 !        defines what subsequent elements on the chain
;1079     2797  1 !        may be returned by PAT$GET_NXT_DUP(0) calls.
;1080     2798  1 !
;1081     2799  1 ! Implicit Outputs:
;1082     2800  1 !       none.
;1083     2801  1 !
;1084     2802  1 ! Return Value:
;1085     2803  1 !
;1086     2804  1 !       0 - when there are no more NTs on the chain,
;1087     2805  1 !       an NT_PTR to the next one, otherwise.
;1088     2806  1 !                                                                                ;
;1089     2807  1 ! Side effects:
;1090     2808  1 !
;1091     2809  1 !       OWN storage for IN_CHAIN and BEGIN_CHAIN get altered.
;1092     2810  1 !--
;1093     2811  1
;1094     2812  2 BEGIN
;1095     2813  2
;1096     2814  2 OWN
;1097     2815  2         IN_CHAIN : REF NT_RECORD,                               ! Keep a static pointer to where we are in
;1098     2816  2                                                                 ! the current hash chain.
;1099     2817  2         BEGIN_CHAIN     : REF NT_RECORD;                        ! If we are somewhere valid in the current
;1100     2818  2                                                                 ! chain, then we also need a static pointer
;1101     2819  2                                                                 ! what was the first element on this chain a
;1102     2820  2                                                                 ! this is the one we get the chain symbol na
;1103     2821  2
```

```
;  1104          2822  2 !++
;  1105          2823  2 ! See if this call marks the beginning of a new hash chain.
;  1106          2824  2 !--
;  1107          2825  3 IF (.INIT_FLAG)
;  1108          2826  2 THEN
;  1109          2827  3         BEGIN
;  1110          2828  3         !++
;  1111          2829  3         ! Set up to be looking at a new chain.
;  1112          2830  3         !--
;  1113          2831  3         BEGIN_CHAIN = .NEW_CHAIN;
;  1114          2832  3         IN_CHAIN = .NEW_CHAIN;
;  1115          2833  3
;  1116          2834  3         !++
;  1117          2835  3         ! We don't define any return value for this type of call.
;  1118          2836  3         !--
;  1119          2837  3         RETURN;
;  1120          2838  2         END;
;  1121          2839  2
;  1122          2840  2 !++
;  1123          2841  2 ! Skip along the chain until the next NT record of interest is found.  We may
;  1124          2842  2 ! already be at the end because of previous calls.
;  1125          2843  2 !--
;  1126          2844  3 IF (.IN_CHAIN EQL 0)
;  1127          2845  2 THEN
;  1128          2846  2         RETURN(0);
;  1129          2847  2
;  1130          2848  2 !++
;  1131          2849  2 ! Otherwise scan the chain.
;  1132          2850  2 !--
;  1133          2851  2 DO
;  1134          2852  3         BEGIN
;  1135          2853  3         !++
;  1136          2854  3         ! Simply check that the symbol names match.
;  1137          2855  3         !--
;  1138          2856  4         IF (CH$EQL( .IN_CHAIN[NT_NAME_CS], IN_CHAIN[NT_NAME_ADDR],
;  1139          2857  4            .BEGIN_CHAIN[NT_NAME_CS], BEGIN_CHAIN[NT_NAME_ADDR])
;  1140          2858  4           )
;  1141          2859  3         THEN
;  1142          2860  4                 BEGIN
;  1143          2861  4                 LOCAL
;  1144          2862  4                         MATCH_NT : REF NT_RECORD;
;  1145          2863  4
;  1146          2864  4                 MATCH_NT = .IN_CHAIN;
;  1147          2865  4                 IN_CHAIN = .IN_CHAIN[NT_FORWARD];
;  1148          2866  4                 RETURN(.MATCH_NT);
;  1149          2867  3                 END;
;  1150          2868  3         END
;  1151          2869  3
;  1152          2870  2 WHILE( (IN_CHAIN = .IN_CHAIN[NT_FORWARD]) NEQ 0 );
;  1153          2871  2 !++
;  1154          2872  2 ! There are no more in the chain to consider.
;  1155          2873  2 !--
;  1156          2874  2 RETURN(0);
;  1157          2875  1 END;
; INFO#212                L1:2832
; Null expression appears in value-required context
```

```
                                                        .PSECT  _PAT$OWN,NOEXE,2

                                                00000 IN_CHAIN:
                                                        .BLKB   4
                                                00004 BEGIN_CHAIN:
                                                        .BLKB   4



                                                        .PSECT  _PAT$CODE,NOWRT,2

                                                007C 00000      .ENTRY  PAT$GET_NXT_DUP, Save R2,R3,R4,R5,R6            ; 2765
                    56 00000000'    EF  9E 00002        MOVAB   IN_CHAIN, R6                                    ;
                    0B      04      AC  E9 00009        BLBC    INIT_FLAG, 1$                                   ; 2825
              04    A6      08      AC  D0 0000D        MOVL    NEW_CHAIN, BEGIN_CHAIN                          ; 2831
                    66      08      AC  D0 00012        MOVL    NEW_CHAIN, IN_CHAIN                             ; 2832
                                    36  11 00016        BRB     4$                                              ; 2827
                                    66  D5 00018 1$:     TSTL    IN_CHAIN                                        ; 2844
                                    32  13 0001A        BEQL    4$                                              ;
                    55 00000000G    EF  D0 0001C 2$:    MOVL    PAT$GL_RST_BEGN, R5                             ; 2856
          54                        66  55 C1 00023     ADDL3   R5, IN_CHAIN, R4                                ;
                    52      0C      A4  9A 00027        MOVZBL  12(R4), R2                                      ;
          50        55      04      A6  C1 0002B        ADDL3   BEGIN_CHAIN, R5, R0                             ; 2857
                    51      0C      A0  9A 00030        MOVZBL  12(R0), R1                                      ;
    51    00        0D      A4      52  2D 00034        CMPC5   R2, 13(R4), #0, R1, 13(R0)                      ;
                            0D      A0      0003A       ;                                                       ;
                                    07  12 0003C        BNEQ    3$                                              ;
                    50              66  D0 0003E        MOVL    IN_CHAIN, MATCH_NT                              ; 2864
                    66              64  3C 00041        MOVZWL  (R4), IN_CHAIN                                  ; 2865
                                        04 00044        RET                                                     ; 2866
          50        66              55  C1 00045 3$:    ADDL3   R5, IN_CHAIN, R0                                ; 2870
                    66              60  3C 00049        MOVZWL  (R0), IN_CHAIN                                  ;
                                    CE  12 0004C        BNEQ    2$                                              ;
                                    50  D4 0004E 4$:     CLRL    R0                                              ; 2875
                                        04 00050        RET                                                     ;

; Routine Size:  81 bytes,    Routine Base:  _PAT$CODE + 0324
```

```
;1159     2876  1 ROUTINE CONCAT_PATHS( DESTINATION, PV_1, PV_2 ) : NOVALUE =
;1160     2877  1
;1161     2878  1 !++
;1162     2879  1 ! Functional Description:
;1163     2880  1 !
;1164     2881  1 !       Concatenate two PATHNAME_VECTORS together and place
;1165     2882  1 !       the resultant vector in the indicated location.
;1166     2883  1 !       If the PV_1 vector is the same as the DESTINATION
;1167     2884  1 !       vector, then this routine is equivalent to routine
;1168     2885  1 !       which just 'tacks' one vector onto the end of another.
;1169     2886  1 !
;1170     2887  1 ! Formal Parameters:
;1171     2888  1 !
;1172     2889  1 !  DESTINATION          -a pointer to where the concatenation
;1173     2890  1 !                        of the PV_1 and PV_2 vectors should be
;1174     2891  1 !                        be stored.
;1175     2892  1 !  PV_1                 -a pointer to the first source vector.  If PV_1[0]
;1176     2893  1 !                        is 0, the PV_1 vector is considered empty.
;1177     2894  1 !  PV_2                 -a pointer to the second source vector.  If PV_2[0]
;1178     2895  1 !                        is 0, the PV_2 vector is considered empty.
;1179     2896  1 !
;1180     2897  1 ! Implicit Inputs:
;1181     2898  1 !
;1182     2899  1 !       None, other than the assumptions made about PATHNAME_VECTORS,
;1183     2900  1 !       namely their characteristics, given via the 'canned' declaration,
;1184     2901  1 !       and the fact that they must end with a 0 entry.
;1185     2902  1 !
;1186     2903  1 ! Implicit Outputs:
;1187     2904  1 !
;1188     2905  1 !       None.
;1189     2906  1 !
;1190     2907  1 ! Routine Value:
;1191     2908  1 !
;1192     2909  1 !       NOVALUE
;1193     2910  1 !
;1194     2911  1 ! Side Effects:
;1195     2912  1 !
;1196     2913  1 !       The two vectors are concatenated.
;1197     2914  1 !--
;1198     2915  1
;1199     2916  2 BEGIN
;1200     2917  2
;1201     2918  2 MAP
;1202     2919  2         DESTINATION : REF PATHNAME_VECTOR,                   ! All three input parameters are pointers to
;1203     2920  2                                                              ! pathname vectors.
;1204     2921  2         PV_1        : REF PATHNAME_VECTOR,
;1205     2922  2         PV_2        : REF PATHNAME_VECTOR;
;1206     2923  2
;1207     2924  2 LOCAL
;1208     2925  2         I_2,                                                 ! Index used for PV_2 vector.
;1209     2926  2         INDEX;                                               ! Used to index thru the destination vector.
;1210     2927  2
;1211     2928  2 !++
;1212     2929  2 ! Initialize an index into the destination and source pathname vectors.
;1213     2930  2 !--
;1214     2931  2 INDEX = 0;
;1215     2932  2 I_2 = 0;
```

```
;  1216          2933  2
;  1217          2934  2 !++
;  1218          2935  2 ! If we were actually given the first vector, copy each element from it into
;  1219          2936  2 ! the destination vector.  We do not consider a 0-pointer vector to be an error;
;  1220          2937  2 ! it just means that that vector doesn't contribute anything to the result.
;  1221          2938  2 !--
;  1222          2939  3 IF (PV_1[0] NEQ 0)
;  1223          2940  2 THEN
;  1224          2941  2         !++
;  1225          2942  2         ! Copy each element, making sure not to overflow.
;  1226          2943  2         !--
;  1227          2944  3         WHILE( .INDEX LSS MAX_PATH_SIZE )
;  1228          2945  2                 DO
;  1229          2946  3                 IF ((DESTINATION[.INDEX] = .PV_1L.INDEX]) EQL 0)
;  1230          2947  2                 THEN
;  1231          2948  2                         !++
;  1232          2949  2                         ! The first 0 element => the end.
;  1233          2950  2                         !--
;  1234          2951  2                         EXITLOOP
;  1235          2952  2                 ELSE
;  1236          2953  2                         INDEX = .INDEX + 1;
;  1237          2954  2 !++
;  1238          2955  2 ! Then add on the second vector, again making sure that we were actually given
;  1239          2956  2 ! something.  Note that we don't have to check overflow of the PV_2 vector
;  1240          2957  2 ! because no matter how long it is, we will overflow the destination vector
;  1241          2958  2 ! before we overflow it.
;  1242          2959  2 !--
;  1243          2960  3 IF (PV_2[0] NEQ 0)
;  1244          2961  2 THEN
;  1245          2962  3         WHILE( .INDEX LSS MAX_PATH_SIZE )
;  1246          2963  2                 DO
;  1247          2964  3                 IF ((DESTINATION[.INDEX] = .PV_2[.I_2]) EQL 0)
;  1248          2965  2                 THEN
;  1249          2966  2                         !++
;  1250          2967  2                         ! This is the only correct way to return from this routine.
;  1251          2968  2                         !--
;  1252          2969  2                         RETURN
;  1253          2970  2                 ELSE
;  1254          2971  3                         BEGIN
;  1255          2972  3                         I_2 = .I_2 +1;
;  1256          2973  3                         INDEX = .INDEX +1;
;  1257          2974  2                         END;
;  1258          2975  2
;  1259          2976  2 !++
;  1260          2977  2 ! If we don't return in the above loop, then it is because we were about
;  1261          2978  2 ! to overflow the destination vector.  We take care of this by effectively wiping
;  1262          2979  2 ! out the vector we have built.
;  1263          2980  2 !--
;  1264          298i  2 DESTINATION[0] = 0;
;  1265          2982  2 RETURN;
;  1266          2983  1 END;


                                0000 00000 CONCAT_PATHS:
```

```
                                                    .WORD   Save nothing                                ; 2876
                           50  7C 00002             CLRQ    INDEX                                       ; 2931
                     08    AC  D5 00004             TSTL    PV_1                                        ; 2939
                           12  13 00007             BEQL    2$                                          ;
               0A          50  D1 00009 1$:         CMPL    INDEX, #10                                  ; 2944
                           0D  18 0000C             BGEQ    2$                                          ;
          04 BC40    08 BC40  D0 0000E              MOVL    @PV_1[INDEX], @DESTINATION[INDEX]           ; 2946
                           04  13 00015             BEQL    2$                                          ;
                           50  D6 00017             INCL    INDEX                                       ; 2953
                           EE  11 00019             BRB     1$                                          ; 2946
                     0C    AC  D5 0001B 2$:         TSTL    PV_2                                        ; 2960
                           14  13 0001E             BEQL    4$                                          ;
               0A          50  D1 00020 3$:         CMPL    INDEX, #10                                  ; 2962
                           0F  18 00023             BGEQ    4$                                          ;
          04 BC40    0C BC41  D0 00025              MOVL    @PV_2[I_2], @DESTINATION[INDEX]             ; 2964
                           09  13 0002C             BEQL    5$                                          ;
                           51  D6 0002E             INCL    I_2                                         ; 2972
                           50  D6 00030             INCL    INDEX                                       ; 2973
                           EC  11 00032             BRB     3$                                          ; 2964
                     04    BC  D4 00034 4$:         CLRL    @DESTINATION                                ; 2981
                           04 00037 5$:             RET                                                 ; 2983

; Routine Size:  56 bytes,    Routine Base:  _PAT$CODE + 0375
```

```
;1268     2984  1 GLOBAL ROUTINE PAT$NT_HASH_FCN( NAME_CS ) =
;1269     2985  1
;1270     2986  1 !++
;1271     2987  1 ! Functional Description:
;1272     2988  1 !
;1273     2989  1 !       Calculate what we call the 'hash' code associated
;1274     2990  1 !       with a given symbol name (counted string).
;1275     2991  1 !       This function localizes the dispersal that we
;1276     2992  1 !       use to distribute symbol NT-records into so-called
;1277     2993  1 !       NT-chains.  Currently this works by simply adding
;1278     2994  1 !       up the characters which make up the symbol name,
;1279     2995  1 !       adding in the count as well, and then truncating
;1280     2996  1 !       the resultant value to one byte so that we can
;1281     2997  1 !       then use this byte to index into the NT Hash Vector
;1282     2998  1 !       that then gives us the pointer to the NT chain for
;1283     2999  1 !       all symbols that 'hash' to this same byte index.
;1284     3000  1 !
;1285     3001  1 ! Formal Parameters:
;1286     3002  1 !
;1287     3003  1 !       NAME_CS -the address of a counted string that
;1288     3004  1 !                is the symbol name we are to 'hash'.
;1289     3005  1 !
;1290     3006  1 ! Implicit Inputs:
;1291     3007  1 !
;1292     3008  1 !       The hash index is supposed to be 1 byte long.
;1293     3009  1 !       (See the literal, NT_HASH_SIZE).
;1294     3010  1 !
;1295     3011  1 ! Implicit Outputs:
;1296     3012  1 !
;1297     3013  1 !       None.
;1298     3014  1 !
;1299     3015  1 ! Routine Value:
;1300     3016  1 !
;1301     3017  1 !       The 1-byte hash index, guaranteed not to exceed
;1302     3018  1 !       the range you can reach with an unsigned byte.
;1303     3019  1 !
;1304     3020  1 ! Side Effects:
;1305     3021  1 !
;1306     3022  1 !       None.
;1307     3023  1 !--
;1308     3024  1
;1309     3025  2 BEGIN
;1310     3026  2
;1311     3027  2 MAP
;1312     3028  2         NAME_CS : CS_POINTER;
;1313     3029  2
;1314     3030  2 LOCAL
;1315     3031  2         TALLY;
;1316     3032  2
;1317     3033  2 !++
;1318     3034  2 ! We simply add up the characters in the supposed name, including the count.
;1319     3035  2 !--
;1320     3036  2 DECR I FROM (TALLY = .NAME_CS[0]) TO 1
;1321     3037  2         DO
;1322     3038  2         TALLY = .TALLY + .NAME_CS[.I];
;1323     3039  2         !++
;1324     3040  2 ! The hash value is later used as a byte index into the NT hash table.
```

```
; 1325          3041  2 .--
; 1326          3042  2 RETURN( .TALLY<0,8,0> );
; 1327          3043  1 END;


                                      0004 00000         .ENTRY  PAT$NT_HASH_FCN, Save R2                        ; 2984
                                51    04    BC   9A 00002        MOVZBL  @NAME_CS, TALLY                         ; 3036
                                50    01    A1   9E 00006        MOVAB   1(R1), I                                ;
                                            08   11 0000A        BRB     2$                                      ;
                                52    04 BC40   9A 0000C 1$:     MOVZBL  @NAME_CS[I], R2                         ; 3038
                                51          52   C0 00011        ADDL2   R2, TALLY                               ;
                                F5          50   F5 00014 2$:     SOBGTR  I, 1$                                   ;
                                50          51   9A 00017        MOVZBL  TALLY, R0                               ; 3042
                                                 04 0001A        RET                                             ; 3043

; Routine Size:  27 bytes,    Routine Base:  _PAT$CODE + 03AD
```

```
;1329     3044  1 ROUTINE PATH_MATCH( PATH_VEC_PTR, NT_PTR ) =
;1330     3045  1
;1331     3046  1 !++
;1332     3047  1 ! Functional Description:
;1333     3048  1 !
;1334     3049  1 !       See if the NT record we are passed a pointer to
;1335     3050  1 !       has the same implicit path name as the one
;1336     3051  1 !       specified in the PATHNAME_VECTOR we are also
;1337     3052  1 !       passed a pointer to.
;1338     3053  1 !
;1339     3054  1 ! Formal Parameters:
;1340     3055  1 !
;1341     3056  1 !  PATH_VEC_PTR         -A pointer to the pathname vector which holds
;1342     3057  1 !                        the path name of the symbol we are to match
;1343     3058  1 !                        with the one implied by the NT scope chain.
;1344     3059  1 !  NT_PTR               -A pointer to the NT record which begins a so-called
;1345     3060  1 !                        scope chain.
;1346     3061  1 !
;1347     3062  1 ! Implicit Inputs:
;1348     3063  1 !
;1349     3064  1 !       -Pathname vectors end with a 0 element; NT scope chains
;1350     3065  1 !        end when PAT$ADD_NT_T_PV says they do.
;1351     3066  1 !       -If the 'top' NT entry in an NT chain IS_GLOBAL,
;1352     3067  1 !        then we assume that a pathname consisting only
;1353     3068  1 !        of that global name is sufficient to 'match' the
;1354     3069  1 !        implicit NT pathname.  This means that PATH_MATCH
;1355     3070  1 !        had better not be called with the user-given pathname
;1356     3071  1 !        (ie nothing has been perpended to it) unless
;1357     3072  1 !        a global answer is acceptable.
;1358     3073  1 !
;1359     3074  1 ! Implicit Outputs:
;1360     3075  1 !
;1361     3076  1 !       None.
;1362     3077  1 !
;1363     3078  1 ! Return Value:
;1364     3079  1 !
;1365     3080  1 !       TRUE, if the paths match,
;1366     3081  1 !       FALSE, otherwise.
;1367     3082  1 !
;1368     3083  1 ! Side Effects:
;1369     3084  1 !
;1370     3085  1 !       None.
;1371     3086  1 !--
;1372     3087  1
;1373     3088  2 BEGIN
;1374     3089  2
;1375     3090  2 MAP
;1376     3091  2         PATH_VEC_PTR : REF PATHNAME_VECTOR,
;1377     3092  2         NT_PTR : REF NT_RECORD;
;1378     3093  2
;1379     3094  2 LOCAL
;1380     3095  2         CS_SRC : CS_POINTER,
;1381     3096  2         CS_DST : CS_POINTER,
;1382     3097  2         PATH_VEC : PATHNAME_VECTOR;
;1383     3098  2
;1384     3099  2 !++
;1385     3100  2 ! Since we want to compare to pathnames which  have a radically different data
```

```
;1386     3101  2 ! structure representation, the simplest thing to do is to convert one of the
;1387     3102  2 ! two different things into the same kind of thing as the other is.  Then
;1388     3103  2 ! comparing them is easy. In our case it is easiest to build a real
;1389     3104  2 ! PATHNAME_VECTOR to correspond to the one implied by the NT scope chain.
;1390     3105  2 !--
;1391     3106  2 PAT$ADD_NT_T_PV( .NT_PTR, PATH_VEC );
;1392     3107  2
;1393     3108  2 !++
;1394     3109  2 ! Now just look thru the two vectors making sure that the CS pointers therein
;1395     3110  2 ! point to identical strings.  Note that we do the comparison up to and
;1396     3111  2 ! including the symbol name which ends the path, since although we know these
;1397     3112  2 ! symbols hash to the same value we don't know that they are identical.
;1398     3113  2 ! Also note that even if we go thru the following loop mathing OK up to and
;1399     3114  2 ! including the MAX_PATH_SIZE'th time, this is still NOT a valid match since
;1400     3115  2 ! we must get a 0 entry to end the pathvectors properly.  This is why we say
;1401     3116  2 ! that falling out of this loop implies failure.
;1402     3117  2 !--
;1403     3118  2 INCR I FROM 0 TO MAX_PATH_SIZE
;1404     3119  2         DO
;1405     3120  3         BEGIN
;1406     3121  3         !++
;1407     3122  3         ! Extract the CS pointers from the pathname vectors and make sure
;1408     3123  3         ! that one of them is not 0 unless the other one is.  If the two paths
;1409     3124  3         ! are to match,  they must end at the same time, otherwise the paths
;1410     3125  3         ! do not match because one is longer than the other.
;1411     3126  3         !--
;1412     3127  3         CS_SRC = .PATH_VEC_PTR[.I];
;1413     3128  3         CS_DST = .PATH_VEC[.I];
;1414     3129  4         IF (.CS_SRC EQL 0) AND (.CS_DST EQL 0)
;1415     3130  3         THEN
;1416     3131  3                 RETURN(TRUE);                                   ! This is one of two places that a match can
;1417     3132  3
;1418     3133  4         IF (.CS_SRC EQL 0) OR (.CS_DST EQL 0)
;1419     3134  3         THEN
;1420     3135  3                 EXITLOOP;                                       ! The pathnames don't match because
;1421     3136  3                                                                 ! one of them is too short.
;1422     3137  3
;1423     3138  3         !++
;1424     3139  3           Now it is safe to actually compare the element strings.
;1425     3140  3         !--
;1426     3141  4         IF (NOT CH$EQL( .CS_SRC[0], CS_SRC[1], .CS_DST[0], CS_DST[1]))
;1427     3142  3         THEN
;1428     3143  3                 EXITLOOP;                                       ! Mismatch because two elements are not the
;1429     3144  2                 END;                                            ! Corresponding pathname elements match.  Go
;1430     3145  2                                                                 ! and check successive elements.
;1431     3146  2
;1432     3147  2 !++
;1433     3148  2 ! If we fall out of the above loop, then the pathnames did not match.
;1434     3149  2 ! As well as a straightforward pathvector match, we must also implement a
;1435     3150  2 ! special match for when the user asks for a global symbol which happens
;1436     3151  2 ! to be a global from a module we have locals for.  The reason for the special
;1437     3152  2 ! case here is because (only) this type of global has a real pathname -
;1438     3153  2 ! namely "mod\glob_name".  While we support the user giving such names, we must
;1439     3154  2 ! also allow him to say simply "glob_name" as well.  To check this, first see
;1440     3155  2 ! that the two symbol names are identical.  Then make sure that this is all
;1441     3156  2 ! of the user-given pathname, and that the corresponding NT record _IS_GLOBAL
;1442     3157  2 ! and is not MODULE.
```

```
;  1443          3158  2 !--
;  1444          3159  2 CS_SRC = .PATH_VEC_PTR[0];
;  1445          3160  3 IF (NOT CH$EQL(.CS_SRC[0], CS_SRC[1], .NT_PTR[NT_NAME_CS], NT_PTR[NT_NAME_ADDR]))
;  1446          3161  2 THEN
;  1447          3162  3         BEGIN
;  1448          3163  3
;  1449          3164  3         RETURN(FALSE);                                  ! Mismatch because two elements are not the
;  1450          3165  2         END;
;  1451          3166  2
;  1452          3167  3 IF (.PATH_VEC_PTR[1] EQL 0 AND .NT_PTR[NT_IS_GLOBAL])
;  1453          3168  2 THEN
;  1454          3169  2         !++
;  1455          3170  2         ! Don't get fooled by the MC_IS_GLOBAL MC record. No module names have
;  1456          3171  2         ! values, so can never be PATH_MATCHed.
;  1457          3172  2         !--
;  1458          3173  3         IF (.NT_PTR[NT_TYPE] NEQ DSC$K_DTYPE_MOD)
;  1459          3174  2         THEN
;  1460          3175  3                 BEGIN
;  1461          3176  3
;  1462          3177  3                         RETURN(TRUE);                   ! This is the second of 2 places were a matc
;  1463          3178  2                 END;
;  1464          3179  2
;  1465          3180  2 RETURN(FALSE);                                          ! The two pathnames do not match.
;  1466          3181  1 END;

                                00FC 00000 PATH_MATCH:
                                           .WORD   Save R2,R3,R4,R5,R6,R7       ; 3044
                          5E      2C   C2 00002    SUBL2   #44, SP              ;
                          5E      DD   00 00005    PUSHL   SP                   ; 3106
                    08    AC      DD   00 00007    PUSHL   NT_PTR               ;
                FC29      CF      02   FB 0000A    CALLS   #2, PAT$ADD_NT_T_PV  ;
                          57  04  AC   D0 0000F    MOVL    PATH_VEC_PTR, R7     ; 3127
                              54   D4   00 00013    CLRL    I                   ;
                          55      6744 D0   00 00015 1$:    MOVL    (R7)[I], CS_SRC     ;
                          56      6E44 D0   00 00019    MOVL    PATH_VEC[I], CS_DST ; 3128
                              50   D4   00 0001D    CLRL    R0                  ; 3129
                              55   D5   00 0001F    TSTL    CS_SRC              ;
                              06   12   00 00021    BNEQ    2$                  ;
                              50   D6   00 00023    INCL    R0                  ;
                              56   D5   00 00025    TSTL    CS_DST              ;
                              48   13   00 00027    BEQL    4$                  ;
                          18      50   E8   00 00029 2$:    BLBS    R0, 3$      ; 3133
                              56   D5   00 0002C    TSTL    CS_DST              ;
                              14   13   00 0002E    BEQL    3$                  ;
                          51      65   9A   00 00030    MOVZBL  (CS_SRC), R1    ; 3141
                          50      66   9A   00 00033    MOVZBL  (CS_DST), R0    ;
50          00      01    A5      51   2D   00 00036    CMPC5   R1, 1(CS_SRC), #0, R0, 1(CS_DST)    ;
                    01    A6           00 0003C    ;
                              04   12   00 0003E    BNEQ    3$                  ;
                D1        54      0A   F3   00 00040    AOBLEQ  #10, I, 1$      ; 3118
                          55      67   D0   00 00044 3$:    MOVL    (R7), CS_SRC    ; 3159
                          51      65   9A   00 00047    MOVZBL  (CS_SRC), R1    ; 3160
                56      08    AC 00000000G EF   C1   00 0004A    ADDL3   PAT$GL_RST_BEGN, NT_PTR, R6    ;
```

```
                50      0C      A6  9A 00053        MOVZBL  12(R6), R0
        50              00      01  A5  51  2D 00057        CMPC5   R1, 1(CS_SRC), #0, R0, 13(R6)
                                        0D  A6      0005D
                                        14  12 0005F        BNEQ    5$
                                        04  A7  D5 00061        TSTL    4(R7)                                   ; 3167
                                        0F  12 00064        BNEQ    5$
                        0B      03      A6  E9 00066        BLBC    3(R6), 5$
                    BC  8F      02      A6  91 0006A        CMPB    2(R6), #188                             ; 3173
                                        04  13 0006F        BEQL    5$
                50              01      D0 00071 4$:        MOVL    #1, R0                                  ; 3177
                                        04 00074        RET
                                        50  D4 00075 5$:        CLRL    R0                                  ; 3181
                                        04 00077        RET

; Routine Size:  120 bytes,    Routine Base:  _PAT$CODE + 03C8
```

```
;1468     3182  1 GLOBAL ROUTINE PAT$VAL_TO_SYM( VALUE, NT_PTR_ADDR, LVT_FLAG ) =
;1469     3183  1
;1470     3184  1 !++
;1471     3185  1 ! Functional Description:
;1472     3186  1 !
;1473     3187  1 !       Implement the search algorithm which PATCH uses to
;1474     3188  1 !       correspond values with symbols.
;1475     3189  1 !
;1476     3190  1 ! Formal Parameters:
;1477     3191  1 !
;1478     3192  1 !       VALUE           -The key we use in the lookup.
;1479     3193  1 !       NT_PTR_ADDR     -A pointer to where we should copy back
;1480     3194  1 !                        the NT_POINTER to the record we find
;1481     3195  1 !                        to correspond to the value.  The
;1482     3196  1 !                        contents of this location are not changed
;1483     3197  1 !                        if no such correspondence is discovered.
;1484     3198  1 !       LVT_FLAG        -Whether or not we should even try for a
;1485     3199  1 !                        match in the literal value table (LVT).
;1486     3200  1 !                        (We always lookup in the SAT first).
;1487     3201  1 ! Warning:
;1488     3202  1 !
;1489     3203  1 !       We return an NT pointer via NT_PTR_ADDR.
;1490     3204  1 !       As long as this returned value is NOT a longword,
;1491     3205  1 !       due to BLISS's inability to use the REF NT_RECORD
;1492     3206  1 !       ONLY as an NT pointer, (i.e. BLISS assumes that
;1493     3207  1 !       REFs are longwords), a caller of this routine should
;1494     3208  1 !       initialize the NT_PTR to 0 before expecting this
;1495     3209  1 !       routine to pass back the value.  This routine
;1496     3210  1 !       can not pass back a longword because we won't know that
;1497     3211  1 !       the address we were passed is not a real NT_PTR
;1498     3212  1 !       field - ie, the 2-bytes that it should be...
;1499     3213  1 !
;1500     3214  1 ! Implicit Inputs:
;1501     3215  1 !
;1502     3216  1 !       None.
;1503     3217  1 !
;1504     3218  1 ! Implicit Outputs:
;1505     3219  1 !
;1506     3220  1 !       The RESULT parameter via NT_PTR_ADDR.
;1507     3221  1 !       (see above.)
;1508     3222  1 !
;1509     3223  1 ! Routine Value:
;1510     3224  1 !
;1511     3225  1 !       TRUE, if a match is found,
;1512     3226  1 !       FALSE, otherwise.
;1513     3227  1 !
;1514     3228  1 ! Side Effects:
;1515     3229  1 !
;1516     3230  1 !
;1517     3231  1 !       The SAT (and LVT) table(s) is/are searched.
;1518     3232  1 !--
;1519     3233  1
;1520     3234  2 BEGIN
;1521     3235  2
;1522     3236  2 MAP
;1523     3237  2         NT_PTR_ADDR : REF RST_POINTER;                  ! The NT pointer we return is an RST-pointer
;1524     3238  2
```

```
;  1525          3239  2 LOCAL
;  1526          3240  2         SAT_PTR : REF SAT_RECORD;
;  1527          3241  2
;  1528          3242  3 IF (VAL_TO_SAT(.VALUE, SAT_PTR))
;  1529          3243  2 THEN
;  1530          3244  3         BEGIN
;  1531          3245  3         !++
;  1532          3246  3         ! Make up for the fact that we are supposed to return an NT pointer,
;  1533          3247  3         ! not the SAT pointer that VAL_TO_SAT gave us.
;  1534          3248  3         !--
;  1535          3249  3         NT_PTR_ADDR[0] = .SAT_PTR[SAT_NT_PTR];
;  1536          3250  3         RETURN(TRUE);
;  1537          3251  2         END;
;  1538          3252  2
;  1539          3253  4 RETURN( IF (.LVT_FLAG)
;  1540          3254  3         THEN
;  1541          3255  3                 LOOKUP_LVT( .VALUE, .NT_PTR_ADDR )
;  1542          3256  3         ELSE
;  1543          3257  2                 FALSE);
;  1544          3258  1 END;


                                                0000 00000          .ENTRY  PAT$VAL_TO_SYM, Save nothing                    ; 3182
                                      5E        04   C2 00002         SUBL2   #4, SP                                          ;
                                                5E   DD 00005         PUSHL   SP                                              ; 3242
                                           04   AC   DD 00007         PUSHL   VALUE                                           ;
                                00000000V  EF        02   FB 0000A         CALLS   #2, VAL_TO_SAT                          ;
                                           09        50   E9 00011         BLBC    R0, 1$                                  ;
                                      08   BC   00   BE   B0 00014         MOVW    @SAT_PTR, @NT_PTR_ADDR                  ; 3249
                                           50        01   D0 00019         MOVL    #1, R0                                  ; 3250
                                                     04   0001C         RET                                             ;
                                           0C   0C   AC   E9 0001D 1$:     BLBC    LVT_FLAG, 2$                            ; 3253
                                           7E   04   AC   7D 00021         MOVQ    VALUE, -(SP)                            ; 3255
                                00000000V  EF        02   FB 00025         CALLS   #2, LOOKUP_LVT                          ;
                                                     04   0002C         RET                                             ;
                                                50   D4   0002D 2$:     CLRL    R0                                      ; 3253
                                                     04   0002F         RET                                             ; 3258

; Routine Size:  48 bytes,    Routine Base:  _PAT$CODE + 0440
```

```
;1546    3259  1 ROUTINE VAL_TO_SAT( VALUE, SAT_PTR_ADDR ) =
;1547    3260  1
;1548    3261  1 !++
;1549    3262  1 ! Functional Description:
;1550    3263  1 !
;1551    3264  1 !       Search the Static Address Table (SAT) for the best
;1552    3265  1 !       match to the given value.
;1553    3266  1 !
;1554    3267  1 ! Formal Parameters:
;1555    3268  1 !
;1556    3269  1 !       VALUE           -The key we use in the lookup.
;1557    3270  1 !       SAT_PTR_ADDR    -A pointer to where we should copy back
;1558    3271  1 !                        the SAT_POINTER to the record we find
;1559    3272  1 !                        to correspond to the value.  The
;1560    3273  1 !                        contents of this location are not changed
;1561    3274  1 !                        if no such correspondence is discovered.
;1562    3275  1 !
;1563    3276  1 ! Implicit Inputs:
;1564    3277  1 !
;1565    3278  1 !       The way we define a 'match' in the SAT is
;1566    3279  1 !       determined solely in this routine.
;1567    3280  1 !
;1568    3281  1 ! Implicit Outputs:
;1569    3282  1 !
;1570    3283  1 !       The RESULT parameter, via SAT_PTR_ADDR, also
;1571    3284  1 !       indicates the corresponding NT record
;1572    3285  1 !       (via SAT_NT_PTR).
;1573    3286  1 !
;1574    3287  1 ! Routine Value:
;1575    3288  1 !
;1576    3289  1 !       TRUE, if a match is found,
;1577    3290  1 !       FALSE, otherwise.
;1578    3291  1 !
;1579    3292  1 ! Side Effects:
;1580    3293  1 !
;1581    3294  1 !       The SAT is searched.
;1582    3295  1 !--
;1583    3296  1
;1584    3297  2 BEGIN
;1585    3298  2
;1586    3299  2 MAP
;1587    3300  2         SAT_PTR_ADDR : REF VECTOR;                      ! The SAT pointer we stuff back is a longwor
;1588    3301  2
;1589    3302  2 LABEL
;1590    3303  2         SEARCH_SAT;
;1591    3304  2
;1592    3305  2 LOCAL
;1593    3306  2         CURRENT_NT  : REF NT_RECORD,                    ! When we look at each record of the SAT_VEC
;1594    3307  2                                                         ! we apply the proper structure.
;1595    3308  2         CURRENT_SAT : REF SAT_RECORD,
;1596    3309  2         NEXT_SAT : REF SAT_RECORD,
;1597    3310  2         BEST_SAT : REF SAT_RECORD;                      ! When we look at each record of the SAT_VEC
;1598    3311  2                                                         ! we apply the proper structure.
;1599    3312  2                                                         ! same time - the 'current' one, and
;1600    3313  2                                                         ! the previous, or so-far 'best', one.
;1601    3314  2
;1602    3315  2 !++
```

```
;  1603     3316  2 ! There starts out being no 'previous' or  best match.  There is also no initial
;  1604     3317  2 ! notion of 'current' sat - we made it different from 'best' to begin with
;  1605     3318  2 ! solely for the diagnostic printout, below.
;  1606     3319  2 !--
;  1607     3320  2 BEST_SAT = 0;
;  1608     3321  2
;  1609     3322  2 !++
;  1610     3323  2 ! Set up to begin the sequential pass of the SAT.
;  1611     3324  2 !--
;  1612     3325  2 PAT$GET_NXT_SAT( SL_ACCE_INIT );
;  1613     3326  2
;  1614     3327  2 !++
;  1615     3328  2 ! Linearly search the SAT looking for an exact match. SAT access type is _RECS
;  1616     3329  2 ! because we want to quit  when the access-mapping routine recognizes that
;  1617     3330  2 ! even though there may be more potential records in the SAT, there are no more
;  1618     3331  2 ! ones currently in use.
;  1619     3332  2 !--
;  1620     3333  3 WHILE( (NEXT_SAT = PAT$GET_NXT_SAT( SL_ACCE_RECS )) NEQ 0 )
;  1621     3334  2         DO
;  1622     3335  2
;  1623     3336  2     SEARCH_SAT:
;  1624     3337  3         BEGIN
;  1625     3338  3
;  1626     3339  3         !++
;  1627     3340  3         ! Update our idea of the current SAT to consider.
;  1628     3341  3         !--
;  1629     3342  3         CURRENT_SAT = .NEXT_SAT;
;  1630     3343  3         CURRENT_NT = .CURRENT_SAT[SAT_NT_PTR];
;  1631     3344  3
;  1632     3345  4         IF (.CURRENT_SAT[SAT_LB] EQL .VALUE)
;  1633     3346  3         THEN
;  1634     3347  4                 BEGIN
;  1635     3348  4                 !++
;  1636     3349  4                 ! We define a match to be 'best' if it is an exact one.  Even
;  1637     3350  4                 ! better, though, is an exact match which is NOT to a p-sect symbol.
;  1638     3351  4                 !--
;  1639     3352  5                 IF (.BEST_SAT EQL 0)
;  1640     3353  4                 THEN
;  1641     3354  5                         BEGIN
;  1642     3355  5                         !++
;  1643     3356  5                         ! Since there were no previous candidates, we take
;  1644     3357  5                         ! the current one as best.
;  1645     3358  5                         !--
;  1646     3359  5                         BEST_SAT = .CURRENT_SAT;
;  1647     3360  5                         END
;  1648     3361  4                 ELSE
;  1649     3362  5                         BEGIN
;  1650     3363  5                         !++
;  1651     3364  5                         ! See if the former 'best' match was exact.
;  1652     3365  5                         !--
;  1653     3366  6                         IF (.BEST_SAT[SAT_LB] NEQ .VALUE)
;  1654     3367  5                         THEN
;  1655     3368  5                                 !++
;  1656     3369  5                                 ! The newer and exact match is preferable to a
;  1657     3370  5                                 ! previous non-exact one no matter what.
;  1658     3371  5                                 !--
;  1659     3372  5                                 BEST_SAT = .CURRENT_SAT
```

```
;  1660          3373  5                     ELSE
;  1661          3374  5                             !++
;  1662          3375  5                             ! Since there was a previous exact match,
;  1663          3376  5                             ! and there now is a current exact match,
;  1664          3377  5                             ! we choose the one which is not a p-sect
;  1665          3378  5                             ! (even if we end up with one which was a
;  1666          3379  5                             !  match only because its UB is 0).
;  1667          3380  5                             !--
;  1668          3381  6                             IF (.CURRENT_NT[NT_TYPE] NEQ DSC$K_DTYPE_PCT)
;  1669          3382  5                             THEN
;  1670          3383  5                                     BEST_SAT = .CURRENT_SAT;
;  1671          3384  4                     END;
;  1672          3385  4
;  1673          3386  4                 !++
;  1674          3387  4                 ! At this point there is nothing further to do
;  1675          3388  4                 ! until we see if there are any more candidates.
;  1676          3389  4                 !--
;  1677          3390  4                 LEAVE SEARCH_SAT;
;  1678          3391  3                 END;
;  1679          3392  3
;  1680          3393  3             !++
;  1681          3394  3             ! Now try for a non-exact match.
;  1682          3395  3             !--
;  1683          3396  4             IF (.CURRENT_SAT[SAT_LB] GTRA .VALUE)
;  1684          3397  3             THEN
;  1685          3398  3                     !++
;  1686          3399  3                     ! Once we get past the value we were
;  1687          3400  3                     ! searching for, we know there will be
;  1688          3401  3                     ! no other candidates because the SAT
;  1689          3402  3                     ! is sorted.
;  1690          3403  3                     !--
;  1691          3404  3                     EXITLOOP;
;  1692          3405  3
;  1693          3406  3             !++
;  1694          3407  3             ! At this point, we know that the CURRENT lower bound! value is strictly
;  1695          3408  3             ! less than VALUE.  Now we are! concerned about the corresponding upper
;  1696          3409  3             ! bound value. Normally this UB value is the address which is the
;  1697          3410  3             ! upper extent to which the corresponding symbol is bound.  Some symbols
;  1698          3411  3             ! don't come with this info, though, so we handle these cases separately.
;  1699          3412  3             !--
;  1700          3413  4             IF (.CURRENT_SAT[SAT_UB] NEQ 0)
;  1701          3414  3             THEN
;  1702          3415  4                     BEGIN
;  1703          3416  4                     !++
;  1704          3417  4                     ! Since we have the UB information, we simply see if the
;  1705          3418  4                     ! CURRENT symbol spans the given VALUE.
;  1706          3419  4                     !--
;  1707          3420  5                     IF (.CURRENT_SAT[SAT_UB] LSSA .VALUE)
;  1708          3421  4                     THEN
;  1709          3422  4                             LEAVE SEARCH_SAT;                       ! This datum ends too soon.
;  1710          3423  4
;  1711          3424  4                     !++
;  1712          3425  4                     ! VALUE falls within this symbol's extent. If we already have
;  1713          3426  4                     ! a match but havn't accepted it as final, then its UB must
;  1714          3427  4                     ! be 0, or its type is  P-sect.  In either case, this new one
;  1715          3428  4                     ! is better because its UB is not 0.
;  1716          3429  4                     !--
```

```
;  1717     3430  4                BEST_SAT = .CURRENT_SAT;
;  1718     3431  4
;  1719     3432  4                !++
;  1720     3433  4                ! If the best one now is of type P-sect, we  should still
;  1721     3434  4                ! look further.
;  1722     3435  4                !--
;  1723     3436  5                IF (.CURRENT_NT[NT_TYPE] NEQ DSC$K_DTYPE_PCT)
;  1724     3437  4                THEN
;  1725     3438  4                        !++
;  1726     3439  4                        ! This is the first non-P-sect match so we'll take it
;  1727     3440  4                        ! without any further checks.  This is where we might
;  1728     3441  4                        ! get into trouble later because there may still be a
;  1729     3442  4                        ! more appropriate match if we were to take SCOPE into
;  1730     3443  4                        ! account.
;  1731     3444  4                        !--
;  1732     3445  4                        EXITLOOP;
;  1733     3446  4
;  1734     3447  4                !++
;  1735     3448  4                ! There is nothing further to consider when we know the UB value.
;  1736     3449  4                !  At this point we want to loop back and perhaps find a
;  1737     3450  4                ! better match.
;  1738     3451  4                !--
;  1739     3452  4                END
;  1740     3453  3          ELSE
;  1741     3454  4                BEGIN
;  1742     3455  4                !++
;  1743     3456  4                ! We must consider it to be a match when the upper bound is 0
;  1744     3457  4                ! because we don't know that it isn't a match.
;  1745     3458  4                ! It is certainly not a good match, though, so we throw it
;  1746     3459  4                ! away unless we don't have anything better.
;  1747     3460  4                !--
;  1748     3461  5                IF (.BEST_SAT EQL 0)
;  1749     3462  4                THEN
;  1750     3463  4                        BEST_SAT = .CURRENT_SAT
;  1751     3464  4                ELSE
;  1752     3465  5                        IF (.BEST_SAT[SAT_UB] EQL 0)
;  1753     3466  4                        THEN
;  1754     3467  4                                BEST_SAT = .CURRENT_SAT
;  1755     3468  4                        ELSE
;  1756     3469  4                                !++
;  1757     3470  4                                ! BEST must be p-sect
;  1758     3471  4                                !--
;  1759     3472  5                                IF (.CURRENT_NT[NT_TYPE] NEQ DSC$K_DTYPE_PCT)                                   ;
;  1760     3473  4                                THEN
;  1761     3474  4                                        BEST_SAT = .CURRENT_SAT;
;  1762     3475  3                END;
;  1763     3476  3
;  1764     3477  2          END;                                                   ! Loop back and try again with the next  SAT
;  1765     3478  2
;  1766     3479  2 !++
;  1767     3480  2 ! If we fall out of the above loop, then there are no more candidates from the
;  1768     3481  2 ! RST to consider.  See if the answer we got was good enough, or if we should
;  1769     3482  2 ! consult the GST.
;  1770     3483  2 !--
;  1771     3484  3 IF (.BEST_SAT EQL 0)
;  1772     3485  2 THEN
;  1773     3486  2          !++
```

```
;  1774          3487  2          ! Any global is better than nothing.
;  1775          3488  2          !--
;  1776          3489  2          BEST_SAT = GBL_VAL_TO_SAT(.VALUE)
;  1777          3490  2 ELSE
;  1778          3491  2          !++
;  1779          3492  2          ! Don't even try for a global if we already got
;  1780          3493  2          ! an exact match from the RST.
;  1781          3494  2          !--
;  1782          3495  3          IF (NOT .VALUE EQLA .BEST_SAT[SAT_LB])
;  1783          3496  2          THEN
;  1784          3497  3                  BEGIN
;  1785          3498  3                  LOCAL
;  1786          3499  3                          BEST_NT : REF NT_RECORD;
;  1787          3500  3
;  1788          3501  3                  !++
;  1789          3502  3                  ! Check for a better match.
;  1790          3503  3                  !--
;  1791          3504  3                  BEST_NT = .BEST_SAT[SAT_NT_PTR];
;  1792          3505  4                  IF (.BEST_SAT[SAT_UB] EQL 0) OR
;  1793          3506  4                      (.BEST_NT[NT_TYPE] EQL DSC$K_DTYPE_PCT))
;  1794          3507  3                  THEN
;  1795          3508  4                          IF ((CURRENT_SAT = GBL_VAL_TO_SAT(.VALUE)) NEQ 0)
;  1796          3509  3                          THEN
;  1797          3510  3                                  !++
;  1798          3511  3                                  ! The global match is better if it is closer.
;  1799          3512  3                                  !--
;  1800          3513  4                                  IF (.CURRENT_SAT[SAT_LB] GTRA .BEST_SAT[SAT_LB])
;  1801          3514  3                                  THEN
;  1802          3515  3                                          BEST_SAT = .CURRENT_SAT;
;  1803          3516  2                      END;
;  1804          3517  2
;  1805          3518  2 !++
;  1806          3519  2 ! Now see how we've done.
;  1807          3520  2 !--
;  1808          3521  3 IF (.BEST_SAT NEQ 0)
;  1809          3522  2 THEN
;  1810          3523  3         BEGIN
;  1811          3524  3         !++
;  1812          3525  3         ! Pass back the NT-pointer to the match and return a success status.
;  1813          3526  3         !--
;  1814          3527  3         SAT_PTR_ADDR[0] = .BEST_SAT;
;  1815          3528  3         RETURN(TRUE);
;  1816          3529  2         END;
;  1817          3530  2
;  1818          3531  2 !++
;  1819          3532  2 ! No 'match' was found - return the standard failure status.
;  1820          3533  2 !--
;  1821          3534  2 RETURN(FALSE);
;  1822          3535  1 END;


                                01FC 00000 VAL_TO_SAT:
                                           .WORD   Save R2,R3,R4,R5,R6,R7,R8                ; 3259
                58 00000000G   EF   9E 00002       MOVAB   PAT$GET_NXT_SAT, R8              ;
```

```
                57     FCC4    CF  9E 00009          MOVAB   GBL_VAL_TO_SAT, R7                       ;
                56 00000000G   EF  9E 0000E          MOVAB   PAT$GL_RST_BEGN, R6                     ;
                               53  D4 00015          CLRL    BEST_SAT                                ; 3320
                               7E  D4 00017          CLRL    -(SP)                                   ; 3325
                68             01  FB 00019          CALLS   #1, PAT$GET_NXT_SAT                     ;
                55         04  AC  D0 0001C          MOVL    VALUE, R5                               ; 3345
                               01  DD 00020 1$:      PUSHL   #1                                      ; 3333
                68             01  FB 00022          CALLS   #1, PAT$GET_NXT_SAT                     ;
                               50  D5 00025          TSTL    NEXT_SAT                                ;
                               4E  13 00027          BEQL    6$                                      ;
                54             50  D0 00029          MOVL    NEXT_SAT, CURRENT_SAT                   ; 3342
                52             64  3C 0002C          MOVZWL  (CURRENT_SAT), CURRENT_NT               ; 3343
                55         02  A4  D1 0002F          CMPL    2(CURRENT_SAT), R5                      ; 3345
                               0C  12 00033          BNEQ    2$                                      ;
                               53  D5 00035          TSTL    BEST_SAT                                ; 3352
                               39  13 00037          BEQL    5$                                      ;
                55         02  A3  D1 00039          CMPL    2(BEST_SAT), R5                         ; 3366
                               28  13 0003D          BEQL    4$                                      ;
                               31  11 0003F          BRB     5$                                      ; 3372
                               34  1A 00041 2$:      BGTRU   6$                                      ; 3396
                           06  A4  D5 00043          TSTL    6(CURRENT_SAT)                          ; 3413
                               16  13 00046          BEQL    3$                                      ;
                55         06  A4  D1 00048          CMPL    6(CURRENT_SAT), R5                      ; 3420
                               D2  1F 0004C          BLSSU   1$                                      ;
                53             54  D0 0004E          MOVL    CURRENT_SAT, BEST_SAT                   ; 3430
          51    52             66  C1 00051          ADDL3   PAT$GL_RST_BEGN, CURRENT_NT, R1         ; 3436
          B8 8F          02  A1  91 00055          CMPB    2(R1), #184                             ;
                               C4  13 0005A          BEQL    1$                                      ;
                               19  11 0005C          BRB     6$                                      ; 3445
                               53  D5 0005E 3$:      TSTL    BEST_SAT                                ; 3461
                               10  13 00060          BEQL    5$                                      ;
                           06  A3  D5 00062          TSTL    6(BEST_SAT)                             ; 3465
                               0B  13 00065          BEQL    5$                                      ;
          51    52             66  C1 00067 4$:      ADDL3   PAT$GL_RST_BEGN, CURRENT_NT, R1         ; 3472
          B8 8F          02  A1  91 0006B          CMPB    2(R1), #184                             ;
                               AE  13 00070          BEQL    1$                                      ;
                53             54  D0 00072 5$:      MOVL    CURRENT_SAT, BEST_SAT                   ; 3474
                               A9  11 00075          BRB     1$                                      ; 3333
                               53  D5 00077 6$:      TSTL    BEST_SAT                                ; 3484
                               0A  12 00079          BNEQ    7$                                      ;
                               55  DD 0007B          PUSHL   R5                                      ; 3489
                67             01  FB 0007D          CALLS   #1, GBL_VAL_TO_SAT                      ;
                53             50  D0 00080          MOVL    R0, BEST_SAT                            ;
                               2C  11 00083          BRB     9$                                      ;
          02 A3                55  D1 00085 7$:      CMPL    R5, 2(BEST_SAT)                         ; 3495
                               26  13 00089          BEQL    9$                                      ;
                50             63  3C 0008B          MOVZWL  (BEST_SAT), BEST_NT                     ; 3504
                           06  A3  D5 0008E          TSTL    6(BEST_SAT)                             ; 3505
                               0A  13 00091          BEQL    8$                                      ;
                50             66  C0 00093          ADDL2   PAT$GL_RST_BEGN, R0                     ; 3506
          B8 8F          02  A0  91 00096          CMPB    2(R0), #184                             ;
                               14  12 0009B          BNEQ    9$                                      ;
                               55  DD 0009D 8$:      PUSHL   R5                                      ; 3508
                67             01  FB 0009F          CALLS   #1, GBL_VAL_TO_SAT                      ;
                54             50  D0 000A2          MOVL    R0, CURRENT_SAT                         ;
                               0A  13 000A5          BEQL    9$                                      ;
          02 A3 02  A4         D1 000A7              CMPL    2(CURRENT_SAT), 2(BEST_SAT)             ; 3513
```

```
                        03  1B 000AC          BLEQU   9$                                                      ;
                53      54  D0 000AE          MOVL    CURRENT_SAT, BEST_SAT                                   ; 3515
                        53  D5 000B1 9$:      TSTL    BEST_SAT                                                ; 3521
                        08  13 000B3          BEQL    10$                                                     ;
            08  BC          53  D0 000B5      MOVL    BEST_SAT, @SAT_PTR_ADDR                                 ; 3527
                50          01  D0 000B9      MOVL    #1, R0                                                  ; 3528
                            04 000BC          RET                                                             ;
                        50  D4 000BD 10$:     CLRL    R0                                                      ; 3534
                            04 000BF          RET                                                             ; 3535

; Routine Size:  192 bytes,    Routine Base:  _PAT$CODE + 0470
```

```
;1824     3536  1 ROUTINE LOOKUP_LVT( VALUE, NT_PTR_ADDR ) =
;1825     3537  1
;1826     3538  1 !++
;1827     3539  1 ! Functional Description:
;1828     3540  1 !
;1829     3541  1 !       Search the literal value table (LVT) for a match
;1830     3542  1 !       to the given value.
;1831     3543  1 !
;1832     3544  1 ! Formal Parameters:
;1833     3545  1 !
;1834     3546  1 !       VALUE           -The key we use in the lookup.
;1835     3547  1 !       NT_PTR_ADDR     -A pointer to where we should copy back
;1836     3548  1 !                        the NT_POINTER to the record we find
;1837     3549  1 !                        to correspond to the value.  The
;1838     3550  1 !                        contents of this location are not changed
;1839     3551  1 !                        if no such correspondence is discovered.
;1840     3552  1 !
;1841     3553  1 ! Warning:
;1842     3554  1 !
;1843     3555  1 !       We return an NT pointer via NT_PTR_ADDR.
;1844     3556  1 !       As long as this returned value is NOT a longword,
;1845     3557  1 !       due to BLISS's inability to use the REF NT_RECORD
;1846     3558  1 !       ONLY as an NT pointer, (i.e. BLISS assumes that
;1847     3559  1 !       REFs are longwords), a caller of this routine should
;1848     3560  1 !       initialize the NT_PTR to 0 before expecting this
;1849     3561  1 !       routine to pass back the value.  This routine
;1850     3562  1 !       can not pass back a longword because we won't know that
;1851     3563  1 !       the address we were passed is not a real NT_PTR
;1852     3564  1 !       field - ie, the 2-bytes that it should be...
;1853     3565  1 !
;1854     3566  1 ! Implicit Inputs:
;1855     3567  1 !
;1856     3568  1 !       The way we define a 'match' in the LVT which is
;1857     3569  1 !       simply that the given value must be exactly the
;1858     3570  1 !       same as the corresponding one from the LVT.  The
;1859     3571  1 !       first one found is always the one passed back.
;1860     3572  1 !
;1861     3573  1 ! Implicit Outputs:
;1862     3574  1 !
;1863     3575  1 !       The RESULT parameter via NT_PTR_ADDR.
;1864     3576  1 !       (see above.)
;1865     3577  1 !
;1866     3578  1 ! Routine Value:
;1867     3579  1 !
;1868     3580  1 !       TRUE, if a match is found,
;1869     3581  1 !       FALSE, otherwise.
;1870     3582  1 !
;1871     3583  1 ! Side Effects:
;1872     3584  1 !
;1873     3585  1 !       The LVT is searched.
;1874     3586  1 !--
;1875     3587  1
;1876     3588  2 BEGIN
;1877     3589  2
;1878     3590  2 MAP
;1879     3591  2         NT_PTR_ADDR : REF RST_POINTER;
;1880     3592  2
```

```
;  1881          3593  2 LOCAL
;  1882          3594  2         LVT_PTR : REF LVT_RECORD;                                   ! When we look at each record of the LVT_VEC
;  1883          3595  2                                                                     ! we apply the proper structure.
;  1884          3596  2
;  1885          3597  2 !++
;  1886          3598  2 ! Set up to begin the sequential pass of the LVT.
;  1887          3599  2 !--
;  1888          3600  2 PAT$GET_NXT_LVT( SL_ACCE_INIT );
;  1889          3601  2
;  1890          3602  2 !++
;  1891          3603  2 ! Linearly search the LVT looking for an exact match. LVT access type is _RECS
;  1892          3604  2 ! because we want to quit  when the access-mapping routine recognizes that
;  1893          3605  2 ! even though there may be more potential records in the LVT, there are no more
;  1894          3606  2 ! ones currently in use.
;  1895          3607  2 !--
;  1896          3608  3 WHILE ((LVT_PTR = PAT$GET_NXT_LVT( SL_ACCE_RECS )) NEQ 0)
;  1897          3609  2         DO
;  1898          3610  3         BEGIN
;  1899          3611  3
;  1900          3612  4         IF (.LVT_PTR[LVT_VALUE] EQL .VALUE)
;  1901          3613  3         THEN
;  1902          3614  4                 BEGIN
;  1903          3615  4                 !++
;  1904          3616  4                 ! This is the only place we can find a match and return an OK status.
;  1905          3617  4                 !--
;  1906          3618  4                 NT_PTR_ADDR[0] = .LVT_PTR[LVT_NT_PTR];
;  1907          3619  4                 RETURN(TRUE);
;  1908          3620  3                 END;
;  1909          3621  3
;  1910          3622  3         !++
;  1911          3623  3         ! Loop back and try again with the next  LVT record.
;  1912          3624  3         !--
;  1913          3625  2         END;
;  1914          3626  2
;  1915          3627  2 !++
;  1916          3628  2 ! If we fall out of the above loop, no match can be found.
;  1917          3629  2 !--
;  1918          3630  2 RETURN(FALSE);
;  1919          3631  1 END;


                                        0004 00000 LOOKUP_LVT:
                                                   .WORD   Save R2                                 ; 3536
                    52 00000000G    EF  9E 00002           MOVAB   PAT$GET_NXT_LVT, R2
                                    7E  D4 00009           CLRL    -(SP)                           ; 3600
                    62              01  FB 0000B           CALLS   #1, PAT$GET_NXT_LVT
                                    01  DD 0000E 1$:       PUSHL   #1                              ; 3608
                    62              01  FB 00010           CALLS   #1, PAT$GET_NXT_LVT
                                    50  D5 00013           TSTL    LVT_PTR
                                    0F  13 00015           BEQL    2$
                04  AC          02 A0  D1 00017           CMPL    2(LVT_PTR), VALUE               ; 3612
                                    F0  12 0001C           BNEQ    1$
                08  BC              60  B0 0001E           MOVW    (LVT_PTR), @NT_PTR_ADDR         ; 3618
                                    50  01  D0 00022       MOVL    #1, R0                          ; 3619
```

```
                        04 00025          RET                                         ;
                50      D4 00026 2$:      CLRL    R0                                  ; 3630
                        04 00028          RET                                         ; 3631

; Routine Size:  41 bytes,    Routine Base:  _PAT$CODE + 0530
```

```
;1921          3632  1 GLOBAL ROUTINE PAT$SYMBOL_VALU( NT_PTR, VALUE_PTR ) =
;1922          3633  1
;1923          3634  1 !++
;1924          3635  1 ! Functional Description:
;1925          3636  1 !
;1926          3637  1 !       -Look up the value associated with a given
;1927          3638  1 !        NT record.
;1928          3639  1 !       -Note that we refer to 'value' associated with a symbol
;1929          3640  1 !        but really mean 'address', since that is as far as RST
;1930          3641  1 !        manipulation handles symbol-value correlation.
;1931          3642  1 !
;1932          3643  1 ! Formal Parameters:
;1933          3644  1 !
;1934          3645  1 !       NT_PTR          -a pointer to the NT_RECORD that corresponds
;1935          3646  1 !                        to the symbol we want the value of.
;1936          3647  1 !       VALUE_PTR       -where we are to copy the value back to.
;1937          3648  1 !
;1938          3649  1 ! Implicit Inputs:
;1939          3650  1 !
;1940          3651  1 !       -We can call PAT$DST_VALUE with a DST_REC_ID.
;1941          3652  1 !        (in BLD's ADD_NT we call it with a DST_RECRD pointer).
;1942          3653  1 !
;1943          3654  1 !       -The value bound to a symbol can be passed
;1944          3655  1 !        back in a longword.
;1945          3656  1 !       -NT records marked NT_IS_GLOBAL may be 1 of 2 types:
;1946          3657  1 !          1) one which was first a normal NT record but
;1947          3658  1 !                which was later marked NT_IS_GLOBAL.  In this case
;1948          3659  1 !                there IS scope info, and we pick up the value
;1949          3660  1 !                from the DST.
;1950          3661  1 !          2) a fake NT record which was created from a GST
;1951          3662  1 !                record.  In this case the 'value' field is stored
;1952          3663  1 !                in the NT_DST_PTR field of the NT record.
;1953          3664  1 !
;1954          3665  1 ! Implicit Outputs:
;1955          3666  1 !
;1956          3667  1 !       None.
;1957          3668  1 !
;1958          3669  1 ! Routine Value:
;1959          3670  1 !
;1960          3671  1 !       TRUE, if the symbol is found successfully and
;1961          3672  1 !             the returned value is the address bound to the symbol,
;1962          3673  1 !       3, if the symbol is found successfully and the value returned is the
;1963          3674  1 !             address of an array descriptor for the symbol,
;1964          3675  1 !       FALSE or 2, otherwise.
;1965          3676  1 !
;1966          3677  1 ! Side Effects:
;1967          3678  1 !
;1968          3679  1 !       The value gets passed back.
;1969          3680  1 !--
;1970          3681  1
;1971          3682  2 BEGIN
;1972          3683  2
;1973          3684  2 MAP
;1974          3685  2         NT_PTR    : REF NT_RECORD,
;1975          3686  2         VALUE_PTR : REF VECTOR[,LONG];
;1976          3687  2
;1977          3688  2 !++
```

```
;  1978          3689  2 ! If this NT record corresponds only to a GLOBAL symbol, there is no associated                ;
;  1979          3690  2 ! DST record.  In this case we use the DST pointer space from the NT record                   ;
;  1980          3691  2 ! to contain the value associated with the global symbol.                                     ;
;  1981          3692  2 !--                                                                                          ;
;  1982          3693  3 IF (.NT_PTR[NT_IS_GLOBAL] AND .NT_PTR[NT_UP_SCOPE] EQL 0)                                    ;
;  1983          3694  2 THEN
;  1984          3695  3         BEGIN
;  1985          3696  3         VALUE_PTR[0] = .NT_PTR[NT_GBL_VALUE];
;  1986          3697  3
;  1987          3698  3         RETURN(TRUE);
;  1988          3699  2         END;
;  1989          3700  2
;  1990          3701  2 !++
;  1991          3702  2 ! We localize all DST understanding to yet another routine.
;  1992          3703  2 !--
;  1993          3704  2 RETURN( PAT$DST_VALUE( .NT_PTR[NT_DST_PTR], .VALUE_PTR));
;  1994          3705  1 END;


                                        0000 00000          .ENTRY  PAT$SYMBOL_VALU, Save nothing                   ; 3632
                50      04  AC 00000000G  EF  C1 00002      ADDL3   PAT$GL_RST_BEGN, NT_PTR, R0                     ; 3693
                        0E          03    A0  E9 00008      BLBC    3(R0), 1$                                       ;
                                    08    A0  B5 0000F      TSTW    8(R0)                                           ;
                                          09  12 00012      BNEQ    1$                                              ;
                        08  BC      04    A0  D0 00014      MOVL    4(R0), @VALUE_PTR                               ; 3696
                        50                01  D0 00019      MOVL    #1, R0                                          ; 3698
                                          04  0001C         RET                                                     ;
                                    08    AC  DD 0001D 1$:  PUSHL   VALUE_PTR                                       ; 3704
                                    04    A0  DD 00020      PUSHL   4(R0)                                           ;
                            00000000V   EF        02  FB 00023      CALLS   #2, PAT$DST_VALUE                       ;
                                                  04  0002A      RET                                             ; 3705

; Routine Size:  43 bytes,    Routine Base:  _PAT$CODE + 0559
```

```
;1996     3706  1 GLOBAL ROUTINE PAT$DST_VALUE( DST_REC_ID, VALUE_PTR ) =
;1997     3707  1
;1998     3708  1 !++
;1999     3709  1 ! Functional Description:
;2000     3710  1 !
;2001     3711  1 !       -Look up the value associated with a given
;2002     3712  1 !        DST record.
;2003     3713  1 !       -Note that we refer to 'value' associated with a symbol
;2004     3714  1 !        but really mean 'address', since that is as far as RST
;2005     3715  1 !        manipulation handles symbol-value correlation.
;2006     3716  1 !        Normally this 'address' is the virtual address which is
;2007     3717  1 !        bound to the symbol.  If the symbol is associated with
;2008     3718  1 !        a descriptor, though, it is the address of the descriptor
;2009     3719  1 !        which is returned for this symbol.
;2010     3720  1 !
;2011     3721  1 ! Formal Parameters:
;2012     3722  1 !
;2013     3723  1 !       DST_REC_ID      -a pointer to the DST RECORD that corresponds
;2014     3724  1 !                        to the symbol we want the value of.
;2015     3725  1 !                       -this may also be the DBGINT-defined DST record ID
;2016     3726  1 !                        which it assigns to each DST record it processes.
;2017     3727  1 !                        ***** see implicit inputs below.
;2018     3728  1 !       VALUE_PTR       -where we are to copy the value back to.
;2019     3729  1 !
;2020     3730  1 ! Implicit Inputs:
;2021     3731  1 !
;2022     3732  1 !       The concept of a DST_REC_ID vs a DST_RECRD pointer
;2023     3733  1 !       is merged by this routine.  i.e. we don't consider
;2024     3734  1 !       these two things to be different even though everything
;2025     3735  1 !       else in PATCH does not preclude this.  If the
;2026     3736  1 !       DST interface routine is changed so that these two things
;2027     3737  1 !       are no longer the same, we must change calls to this routine
;2028     3738  1 !       to be consistent.  So far this routine is called
;2029     3739  1 !       by PAT$SYMBOL_VALU and ADD_NT only.
;2030     3740  1 !
;2031     3741  1 !       -The value bound to a symbol can be passed
;2032     3742  1 !        back in a longword.
;2033     3743  1 !
;2034     3744  1 ! Implicit Outputs:
;2035     3745  1 !
;2036     3746  1 !       None.
;2037     3747  1 !
;2038     3748  1 ! Routine Value:
;2039     3749  1 !
;2040     3750  1 !       TRUE, if the symbol is found successfully and the value returned is the
;2041     3751  1 !             address bound to the symbol,
;2042     3752  1 !       3, if the symbol is found successfully and the value returned is the
;2043     3753  1 !             address of an array descriptor for the symbol,
;2044     3754  1 !       FALSE or 2, otherwise.
;2045     3755  1 !
;2046     3756  1 ! Side Effects:
;2047     3757  1 !
;2048     3758  1 !       The value gets returned.
;2049     3759  1 !
;2050     3760  1 !       The DST is 'read' in such a way as to ensure that
;2051     3761  1 !       if any PAT$GET_NXT_DST sequence is in progress it
;2052     3762  1 !       is NOT disturbed by this call.
```

```
;2053     3763  1 !--
;2054     3764  1
;2055     3765  2 BEGIN
;2056     3766  2
;2057     3767  2 MAP
;2058     3768  2         VALUE_PTR : REF VECTOR[,LONG];
;2059     3769  2
;2060     3770  2 LOCAL
;2061     3771  2         STATUS,                                         ! RETURNED STATUS CODE
;2062     3772  2         MAPPED_ADR : REF BLOCK[,BYTE],                  ! Mapped descriptor address
;2063     3773  2         ISE_ADR,                                        ! Image Section Address
;2064     3774  2         VALUE,                                          ! Used to accumulate the symbol's value.
;2065     3775  2         ACCESS,                                         ! The access field from the DST record.
;2066     3776  2         DST_RECRD : REF DST_RECORD;
;2067     3777  2
;2068     3778  2 !++
;2069     3779  2 ! Fetch the indicated DST record.
;2070     3780  2 !--
;2071     3781  2 STATUS = TRUE;                                          ! ASSUME EVERYTHING WORKS
;2072     3782  3 IF ((DST_RECRD = PAT$GET_DST_REC( .DST_REC_ID )) EQL 0)
;2073     3783  2 THEN
;2074     3784  2         !++
;2075     3785  2         ! The supposed record does not exist.  An error message should already
;2076     3786  2         ! have been produced.
;2077     3787  2         !--
;2078     3788  2         RETURN(FALSE);
;2079     3789  2
;2080     3790  2 !++
;2081     3791  2 ! How we pick up the value depends on what class of DST record this is.
;2082     3792  2 !--
;2083     3793  3 IF (.DST_RECRD[DSTR_TYPE] EQL DSC$K_DTYPE_Z)
;2084     3794  2 THEN
;2085     3795  3         BEGIN
;2086     3796  3         !++
;2087     3797  3         ! BLISS Type Zero records have a format which is different from
;2088     3798  3         ! standard DST records.
;2089     3799  3         !--
;2090     3800  3         BIND
;2091     3801  3                 BLZ_RECRD = DST_RECRD : REF BLZ_RECORD;
;2092     3802  3
;2093     3803  3         !++
;2094     3804  3         ! We can't make any sense out of these records (yet) unless
;2095     3805  3         !       1) no optional type into was given
;2096     3806  3         !          (i.e. we only get the standard 3 bytes)
;2097     3807  3         !       2) the structure attribute is 0
;2098     3808  3         !       3) the sub-type is within the range we currently support.
;2099     3809  3         !          (i.e. only FORMALs and SYMBOLs are OK)
;2100     3810  3         !
;2101     3811  3         ! We have already implicitly checked the first two of these because we
;2102     3812  3         ! wouldn't have entered the symbol into the NT had this check
;2103     3813  3         ! not succeeded.
;2104     3814  3         !--
;2105     3815  3
;2106     3816  3         !++
;2107     3817  3         ! Pick up the value of this symbol.
;2108     3818  3         !--
;2109     3819  3         STATUS = STD_SYM_EVAL(VALUE, .BLZ_RECRD[BLZ_ACCESS], .BLZ_RECRD[BLZ_VALUE]);
```

```
;  2110          3820  3
;  2111          3821  3          !++
;  2112          3822  3          ! End of special handling for type ZERO variables.
;  2113          3823  3          !--
;  2114          3824  3          END
;  2115          3825  2 ELSE
;  2116          3826  2          !++
;  2117          3827  2          ! Class 1 is the so-called SRM 'standard' types.
;  2118          3828  2          !--
;  2119          3829  3          IF (.DST_RECRD[DSTR_TYPE] LEQ DST_TYP_HIGHEST)
;  2120          3830  2          THEN
;  2121          3831  3                  BEGIN
;  2122          3832  4                  IF (.DST_RECRD[DSTR_ACCES_TYPE] EQL ACCS_DESCRIPTOR)
;  2123          3833  3                  THEN
;  2124          3834  4                          BEGIN
;  2125          3835  4                          !++
;  2126          3836  4                          ! Ignore dynamically-located array descriptors.  This
;  2127          3837  4                          ! means that only the so-called "PC" relative ones are
;  2128          3838  4                          ! supported.
;  2129          3839  4                          !--
;  2130          3840  4                          IF (.DST_RECRD[DSTR_ACCES_BASD] NEQ 2) OR
;  2131          3841  5                             (.DST_RECRD[DSTR_ACCES_BREG] NEQ 15)
;  2132          3842  4                          THEN
;  2133          3843  4                                  RETURN(FALSE);
;  2134          3844  4
;  2135          3845  4                          !++
;  2136          3846  4                          ! The descriptor is in the DST record itself.
;  2137          3847  4                          ! The notion of PC is as though the 'DST' PC
;  2138          3848  4                          ! were just used to 'pick up' the _VALUE field,
;  2139          3849  4                          ! which gives how far from there to displace
;  2140          3850  4                          ! to find the actual array descriptor.
;  2141          3851  4                          ! For the routine PAT$SYM_TO_VAL, the status
;  2142          3852  4                          ! code returned must indicate that this value is
;  2143          3853  4                          ! an array descriptor address.
;  2144          3854  4                          !--
;  2145          3855  4                          VALUE = .DST_RECRD[DSTR_VALUE];
;  2146          3856  4                          VALUE = .VALUE + DST_RECRD[DSTR_VALUE] + %UPVAL;
;  2147          3857  4                          STATUS = 3;
;  2148          3858  4                          END
;  2149          3859  3                  ELSE
;  2150          3860  3                          !++
;  2151          3861  3                          ! For standard types, we simply apply the corresponding
;  2152          3862  3                          ! standard algorithm.  The various TYPEs will be used
;  2153          3863  3                          ! mainly to sort out what we do with the value
;  2154          3864  3                          ! after the algorithm tells us how to get it, but
;  2155          3865  3                          ! this is handled outside of this routine.
;  2156          3866  3                          !--
;  2157          3867  3                          STATUS = STD_SYM_EVAL(VALUE, .DST_RECRD[DSTR_ACCESS], .DST_RECRD[DSTR_VALUE]);
;  2158          3868  3                  END
;  2159          3869  2          ELSE
;  2160          3870  3                  BEGIN
;  2161          3871  3                  !++
;  2162          3872  3                  ! Class 2 is 'the rest'.  We expect them to be from the standard
;  2163          3873  3                  ! DSTR_TYPEs class.
;  2164          3874  3                  !--
;  2165          3875  3                  CASE .DST_RECRD[DSTR_TYPE] FROM DST_DST_LOWEST TO DST_DST_HIGHEST OF
;  2166          3876  3                      SET
```

```
;  2167          3877  3
;  2168          3878  3                    [DSC$K_DTYPE_FLD]:  ! Bliss fields.
;  2169          3879  3
;  2170          3880  4                        BEGIN
;  2171          3881  4                        $FAO_TT_OUT('!/FLD not supported');
;  2172          3882  4                        RETURN(FALSE);
;  2173          3883  3                        END;
;  2174          3884  3
;  2175          3885  3                    [DSC$K_DTYPE_PCT,
;  2176          3886  3                     DSC$K_DTYPE_LBL,
;  2177          3887  3                     dsc$k_dtype_slb,
;  2178          3888  3                     DSC$K_DTYPE_RTN]:  ! These ones have values.
;  2179          3889  3
;  2180          3890  3                        VALUE = .DST_RECRD[DSTR_VALUE];
;  2181          3891  3
;  2182          3892  3                    [INRANGE,OUTRANGE]: ! Probably an error.
;  2183          3893  3
;  2184          3894  3                        RETURN(FALSE);
;  2185          3895  3
;  2186          3896  3                    TES;
;  2187          3897  3
;  2188          3898  2                END;
;  2189          3899  2
;  2190          3900  2 !++
;  2191          3901  2 ! At this point, the value has been obtained or a RETURN has been done.
;  2192          3902  2 ! We simply pass back this value and return a good status code.
;  2193          3903  2 !--
;  2194          3904  2 VALUE_PTR[0] = .VALUE;
;  2195          3905  2 RETURN(.STATUS);
;  2196          3906  1 END;
; INFO#212                 L1:3881
; Null expression appears in value-required context


                                                                .PSECT  _PAT$PLIT,NOWRT,NOEXE,0

                                                        13  00000 P.AAA:   .BYTE   19
6F  70  70  75  73  20  74  6F  6E  20  44  4C  46  2F  21  00001           .ASCII  \!/FLD not supported\                   ;
                                            64  65  74  72  00010                                                       ;


                                                                .PSECT  _PAT$CODE,NOWRT,2

                                                      000C  00000           .ENTRY  PAT$DST_VALUE, Save R2,R3               ; 3706
                                        5E              04   C2  00002      SUBL2   #4, SP                                  ;
                                        53              01   D0  00005      MOVL    #1, STATUS                              ; 3781
                                                    04  AC   DD  00008      PUSHL   DST_REC_ID                              ; 3782
                                00000000G       EF      01   FB  0000B      CALLS   #1, PAT$GET_DST_REC                     ;
                                        52              50   D0  00012      MOVL    R0, DST_RECRD                           ;
                                                        7E   13  00015      BEQL    7$                                      ;
                                        50          01  A2   9A  00017      MOVZBL  1(DST_RECRD), R0                        ; 3793
                                                        09   12  0001B      BNEQ    1$                                      ;
                                                    06  A2   DD  0001D      PUSHL   6(DST_RECRD)                            ; 3819
                                        7E          04  A2   9A  00020      MOVZBL  4(DST_RECRD), -(SP)                     ;
                                                        38   11  00024      BRB     3$                                      ;
```

```
                                 17              50  91 00026 1$:     CMPB    R0, #23                                 ; 3829
                                                 42  1A 00029       BGTRU   4$
          02      02   A2            02          00  ED 0002B       CMPZV   #0, #2, 2(DST_RECRD), #2                ; 3832
                                                 24  12 00031       BNEQ    2$
          02      02   A2            02          02  ED 00033       CMPZV   #2, #2, 2(DST_RECRD), #2                ; 3840
                                                 68  12 00039       BNEQ    10$
          0F      02   A2            04          04  ED 0003B       CMPZV   #4, #4, 2(DST_RECRD), #15               ; 3841
                                                 60  12 00041       BNEQ    10$
                                      6E    03    A2  D0 00043       MOVL    3(DST_RECRD), VALUE                     ; 3855
                                      50    03    A2  9E 00047       MOVAB   3(DST_RECRD), R0                        ; 3856
                                      50          6E  C0 0004B       ADDL2   VALUE, R0
                                      6E    04    A0  9E 0004E       MOVAB   4(R0), VALUE
                                      53          03  D0 00052       MOVL    #3, STATUS                              ; 3857
                                                 44  11 00055       BRB     9$                                      ; 3832
                                      03    A2        DD 00057 2$:     PUSHL   3(DST_RECRD)                            ; 3867
                                      7E    02    A2  98 0005A       CVTBL   2(DST_RECRD), -(SP)
                                      08    AE  9F 0005E 3$:     PUSHAB  VALUE
                00000000V             EF          03  FB 00061       CALLS   #3, STD_SYM_EVAL
                                      53          50  D0 00068       MOVL    R0, STATUS
                                                 2E  11 0006B       BRB     9$                                      ; 3829          ; R
                    08      B7    8F              50  8F 0006D 4$:     CASEB   R0, #183, #8                            ; 3875
          0025          0031          0025          0014      00072 5$:     .WORD   6$-5$,-
          0025          0031          0031          0025      0007A           8$-5$,-
                                                    0031      00082           10$-5$,-
                                                                              8$-5$,-
                                                                              8$-5$,-
                                                                              10$-5$,-
                                                                              10$-5$,-
                                                                              8$-5$,-
                                                                              10$-5$
                                                 1D  11 00084       BRB     10$                                     ; 3894
                                                 7E  D4 00086 6$:     CLRL    -(SP)                                   ; 3881
                                00000000'        EF  9F 00088       PUSHAB  P.AAA
                        00000000G   EF                  02  FB 0008E       CALLS   #2, PAT$FAO_OUT
                                                 0C  11 00095 7$:     BRB     10$                                     ; 3882
                                      6E    03    A2  D0 00097 8$:     MOVL    3(DST_RECRD), VALUE                     ; 3890
                                08    BC          6E  D0 0009B 9$:     MOVL    VALUE, @VALUE_PTR                       ; 3904
                                      50          53  D0 0009F       MOVL    STATUS, R0                              ; 3905
                                                 04 000A2       RET
                                                 50  D4 000A3 10$:     CLRL    R0                                      ; 3906
                                                 04 000A5       RET

; Routine Size:  166 bytes,    Routine Base:  _PAT$CODE + 0584
```

```
;  2198          3907  1 ROUTINE STD_SYM_EVAL( PASS_BACK_ADDR, ACCESS, IN_VALUE ) =
;  2199          3908  1
;  2200          3909  1 !++
;  2201          3910  1 ! Functional Description:
;  2202          3911  1 !
;  2203          3912  1 !       Implement the algorithm which comes up with
;  2204          3913  1 !       a symbol's value given the so-called ACCESS
;  2205          3914  1 !       and IN_VALUE fields taken from (various places within)
;  2206          3915  1 !       the DST record for a symbol which uses
;  2207          3916  1 !       'standard encoding'.
;  2208          3917  1 !
;  2209          3918  1 !       See CP0021.MEM, pgs 9-10 for this algorithm.
;  2210          3919  1 !
;  2211          3920  1 ! Formal Parameters:
;  2212          3921  1 !
;  2213          3922  1 !       PASS_BACK_ADDR  -The address of where we stuff back the value   -LT
;  2214          3923  1 !                        the algorithm determines.
;  2215          3924  1 !       ACCESS          -The 1-byte field which encodes the
;  2216          3925  1 !                        3-element fields described in CP0021.MEM
;  2217          3926  1 !                        for so-called 'standard encoding'.
;  2218          3927  1 !       IN_VALUE        -The value field which may be used in
;  2219          3928  1 !                        conjunction with ACCESS.
;  2220          3929  1 !
;  2221          3930  1 ! Implicit Inputs:
;  2222          3931  1 !
;  2223          3932  1 !       That the value bound to a symbol can be passed
;  2224          3933  1 !       back in a longword.
;  2225          3934  1 !
;  2226          3935  1 ! Implicit Outputs:
;  2227          3936  1 !
;  2228          3937  1 !       The value which is currently associated with the symbol which
;  2229          3938  1 !       presumably corresponds to the ACCESS and IN_VALUE fields given,
;  2230          3939  1 !       is passed back.
;  2231          3940  1 !
;  2232          3941  1 ! Routine Value:
;  2233          3942  1 !
;  2234          3943  1 !       TRUE, if all goes OK and the value gets passed back.
;  2235          3944  1 !       NOT TRUE, otherwise.  In this case we distinguish two possibilities:
;  2236          3945  1 !                       0 => a real error - the symbol could not be evaluated.
;  2237          3946  1 !               and     2 => a soft error - the evaluation failed because of an
;  2238          3947  1 !                                           inappropriate context.
;  2239          3948  1 !
;  2240          3949  1 ! Side Effects:
;  2241          3950  1 !
;  2242          3951  1 !       The value which is currently associated with
;  2243          3952  1 !       the symbol which presumably corresponds to
;  2244          3953  1 !       the ACCESS and IN_VALUE fields given.
;  2245          3954  1 !--
;  2246          3955  1
;  2247          3956  2 BEGIN
;  2248          3957  2
;  2249          3958  2 MAP
;  2250          3959  2         PASS_BACK_ADDR : REF VECTOR;                    ! THE VALUE WE PASS BACK IS A LONGWORD
;  2251          3960  2
;  2252          3961  2 LOCAL
;  2253          3962  2         RET_VALUE;                                      ! We accumulate the value we return.
;  2254          3963  2
```

```
;  2255          3964  2 !++
;  2256          3965  2 ! ******* THIS ROUTINE MUST BE WRITTEN WHEN BLISS VARIABLES ARE ADDED. *******
;  2257          3966  2 !--
;  2258          3967  2 RET_VALUE = .IN_VALUE;
;  2259          3968  3 IF (.ACCESS<2,1,0>)                                          ! If "indirect" set
;  2260          3969  2 THEN
;  2261          3970  2 ! ******** WARNING:  Don't know if going indirect is to a mapped or
;  2262          3971  2 !                      non-mapped address.  Therfore return error for now.
;  2263          3972  2 !       RET_VALUE = .(.RET_VALUE);
;  2264          3973  2 !       RETURN(2);
;  2265          3974  2 PASS_BACK_ADDR[0] = .RET_VALUE;
;  2266          3975  2 RETURN(TRUE);
;  2267          3976  1 END;


                                        0000 00000 STD_SYM_EVAL:
                                                   .WORD   Save nothing                          ; 3907
                        50      0C      AC  D0 00002       MOVL    IN_VALUE, RET_VALUE          ; 3967
                04      08      AC      02  E1 00006       BBC     #2, ACCESS, 1$               ; 3968
                        50              02  D0 0000B       MOVL    #2, R0                       ; 3973
                                        04  0000E       RET                                  ;
                        04      BC      50  D0 0000F 1$:   MOVL    RET_VALUE, @PASS_BACK_ADDR   ; 3974
                        50              01  D0 00013       MOVL    #1, R0                       ; 3975
                                        04  00016       RET                                  ; 3976

; Routine Size:  23 bytes,    Routine Base:  _PAT$CODE + 062A
```

```
; 2269          3977  1 END
; 2270          3978  0 ELUDOM                          . End of module


;                                  PSECT SUMMARY

;       Name                    Bytes                           Attributes

;  _PAT$CODE                     1601  NOVEC,NOWRT,  RD ,  EXE,NOSHR,  LCL,  REL,  CON,NOPIC,ALIGN(2)
;  _PAT$OWN                         8  NOVEC,  WRT,  RD ,NOEXE,NOSHR,  LCL,  REL,  CON,NOPIC,ALIGN(2)
;  _PAT$PLIT                       20  NOVEC,NOWRT,  RD ,NOEXE,NOSHR,  LCL,  REL,  CON,NOPIC,ALIGN(0)


;                               Library Statistics

;                                      -------- Symbols --------      Pages       Processing
;       File                           Total    Loaded   Percent      Mapped      Time

;  _$255$DUA28:[SYSLIB]LIB.L32;1        18619        14         0      1000          00:01.8


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


;                                       COMMAND QUALIFIERS

;       BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/VARIANT:1/LIS=LIS$:PATRST/OBJ=OBJ$:PATRST MSRC$:PATRST/UPDATE=(ENH$:PATRST)

; Size:          1601 code + 28 data bytes
; Run Time:        00:49.0
; Elapsed Time:    02:30.8
; Lines/CPU Min:    4874
; Lexemes/CPU-Min: 26019
; Memory Used:  207 pages
; Compilation Complete
```

PATREB
LIS

PATSCA
LIS

PATSIO
LIS

PATRST
LIS

PATSPA
LIS

PATSSV
LIS