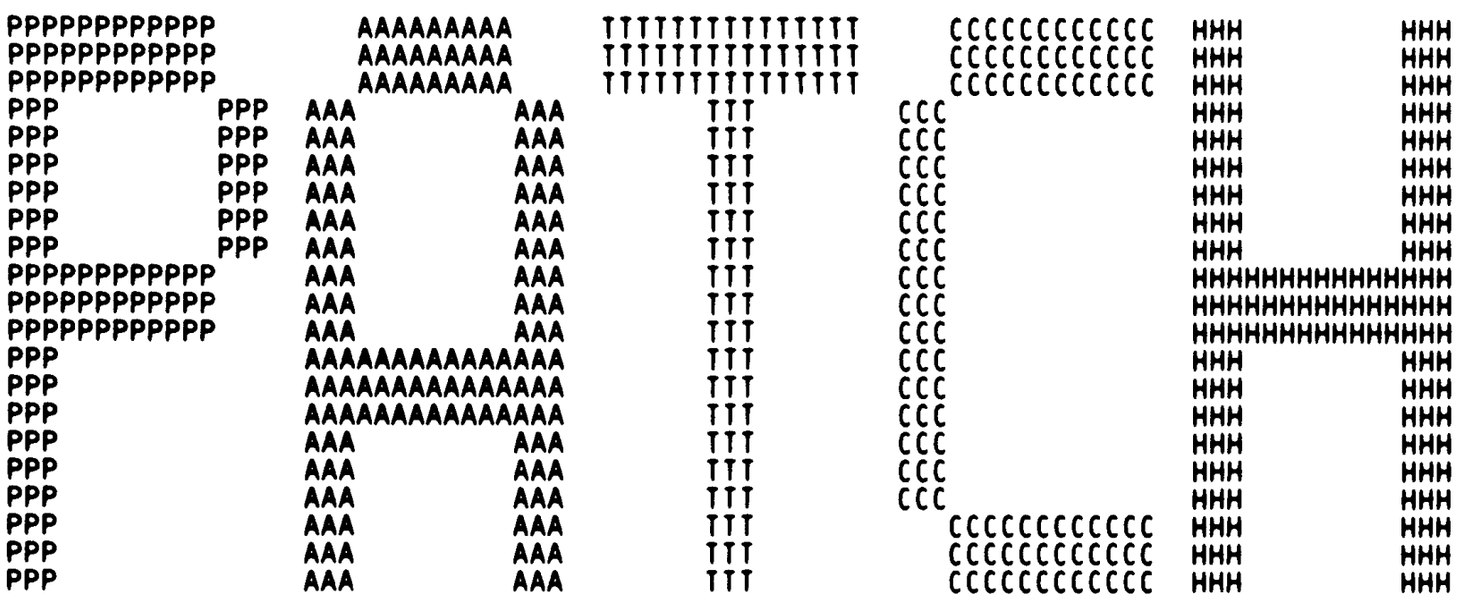
PPPPPPPPPPPP AAAAAAAAA TTTTTTTTTTTTTTT CCCCCCCCCCCC HHH HHH
PPP PPP AAA AAA TTT CCC HHH HHH
PPPPPPPPPPPP AAAAAAAAAAAAAAAA TTT CCC HHHHHHHHHHHHHHHH
PPP AAA AAA TTT CCC HHH HHH
PPP AAA AAA TTT CCCCCCCCCCCC HHH HHH

FILEID**PATBLD

```
PPPPPPPP      AAAAAA    TTTTTTTTTT  BBBBBBBB    LL          DDDDDDDD
PPPPPPPP      AAAAAA    TTTTTTTTTT  BBBBBBBB    LL          DDDDDDDD
PP       PP  AA      AA      TT      BB      BB  LL          DD      DD
PP       PP  AA      AA      TT      BB      BB  LL          DD      DD
PP       PP  AA      AA      TT      BB      BB  LL          DD      DD
PP       PP  AA      AA      TT      BB      BB  LL          DD      DD
PPPPPPPP     AA      AA      TT      BBBBBBBB    LL          DD      DD
PPPPPPPP     AA      AA      TT      BBBBBBBB    LL          DD      DD
PP           AAAAAAAAAA      TT      BB      BB  LL          DD      DD
PP           AAAAAAAAAA      TT      BB      BB  LL          DD      DD
PP           AA      AA      TT      BB      BB  LL          DD      DD      ....
PP           AA      AA      TT      BB      BB  LL          DD      DD      ....
PP           AA      AA      TT      BBBBBBBB    LLLLLLLLLL  DDDDDDDD        ....
PP           AA      AA      TT      BBBBBBBB    LLLLLLLLLL  DDDDDDDD        ....

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 PATBLD ( !
;     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 !++
;    35          0035  1 ! FACILITY:     PATCH
;    36          0036  1 !
;    37          0037  1 ! ABSTRACT:     Initialize and build the Runtime Symbol Table (RST) data structures.
;    38          0038  1 !
;    39          0039  1 !
;    40          0040  1 ! ENVIRONMENT:  This module runs on VAX under STARLET, user mode, non-AST level.
;    41          0041  1 !
;    42          0042  1 ! Author:       Kevin Pammett, August 12, 1977.
;    43          0043  1 !
;    44          0044  1 !
;    45          0045  1 ! Modified by:
;    46          0046  1 !
;    47          0047  1 !       V03-003 MCN0151         Maria del C. Nasr       13-Feb-1984
;    48          0048  1 !               Make changes to eliminate compiler informational messages.
;    49          0049  1 !
;    50          0050  1 !       V03-002 MTR0012         Mike Rhodes             16-Aug-1982
;    51          0051  1 !               Modify file names to remove duplicate file name useage
;    52          0052  1 !               between code and require files.
;    53          0053  1 !
;    54          0054  1 !       V03-001 MTR0010         Mike Rhodes             09-Jul-1982
;    55          0055  1 !               Fix empty compund expressions to have a value of TRUE.
;    56          0056  1 !
;    57          0057  1 !       V02-018 PCG0001         Peter George            02-FEB-1981
```

```
;    58          0058  1 !                Add require statement for LIB$:PATDEF.REQ
;    59          0059  1 !
;    60          0060  1 !       V0117   CNH0023         Chris Hume              16-Nov-1979     14:00
;    61          0061  1 !               Turn off ISD$V_LASTCLU for all ISD's when PATCH Area is added
;    62          0062  1 !               to an image.  Also unrecognized languages will now be processed
;    63          0063  1 !               as though they were MACRO.
;    64          0064  1 !               ([PATCH]PATMAI.B32 02.28, PATSPA.B32 01.05)
;    65          0065  1 !
;    66          0066  1 ! MODIFICATIONS:
;    67          0067  1 !
;    68          0068  1 !  NO   DATE            PROGRAMMER          PURPOSE
;    69          0069  1 !  --   ----            ----------          -------
;    70          0070  1 !
;    71          0071  1 !  00   19-DEC-77       K.D. MORSE          ADAPT VERSION 36 FOR PATCH.
;    72          0072  1 !  01   2-JAN-78        K.D. MORSE          MINOR CHANGES IN
;    73          0073  1 !                                           CASE THERE IS NO GST.
;    74          0074  1 !  02   4-JAN-78        K.D. MORSE          FIX BUG THAT COMPUTES LENGTH
;    75          0075  1 !                                           OF TYPED VARIABLE.
;    76          0076  1 !  03   4-JAN-78        K.D. MORSE          PAT$DST_VALUE NOW DISTINGUISHES
;    77          0077  1 !                                           BETWEEN EVALUATE ERRORS. (37)
;    78          0078  1 !                                           FIX BUG IN ADD_MODULE BY SETTING
;    79          0079  1 !                                           THE NT_IS_BOUNDED BIT ONCE THE
;    80          0080  1 !                                           SYMBOL IS BOUNDED. (38)
;    81          0081  1 !                                           NO CHANGES FOR VERS 39.
;    82          0082  1 !  04   11-JAN-78       K.D. MORSE          ADD CHECKS FOR BIT FIELDS IN
;    83          0083  1 !                                           SECOND TYPE BYTE IN DST,
;    84          0084  1 !                                           (DSTR_ACC01 AND DSTR_ACC23).
;    85          0085  1 !  05   24-JAN-78       K.D. MORSE          NO CHANGES FOR 39-40.
;    86          0086  1 !  06   28-FEB-78       K.D. MORSE          CHANGE BUILD_MC TO IGNORE THE
;    87          0087  1 !                                           REST OF THE DST IF >50% OF THE
;    88          0088  1 !                                           RST SPACE HAS BEEN TAKEN UP FOR
;    89          0089  1 !                                           MC'S ONLY. (41)
;    90          0090  1 !  07   0'-MAR-78       K.D. MORSE          NO CHANGES FOR 42-43, PATCH
;    91          0091  1 !                                           WILL NOT ACCEPT FORTRAN ARRAYS
;    92          0092  1 !                                           AS INPUT AND SO SHOULD NOT
;    93          0093  1 !                                           OUTPUT THEM.
;    94          0094  1 !  08   24-MAR-78       K.D. MORSE          NO CHANGES FOR 44-45.
;    95          0095  1 !  09   28-MAR-78       K.D. MORSE          ADD CODE TO ACCEPT ARRAYS AND
;    96          0096  1 !                                           "PC" RELATIVE DESCRIPTORS.
;    97          0097  1 !  10   07-APR-78       K.D. MORSE          REWORKED GLOBALS SO THAT THEY
;    98          0098  1 !                                           ARE NOW MAPPED INSTEAD OF IN
;    99          0099  1 !                                           THE RST.  (46) REMOVED CODE IN
;   100          0100  1 !                                           ADD_SAT THAT CHECKED IF A
;   101          0101  1 !                                           SYMBOL WAS PREVIOUSLY THERE (47).
;   102          0102  1 !                                           MC_GBL_LOCKED IS INITIALIZED (48).
;   103          0103  1 !                                           NO CHANGES FOR VERS 49
;   104          0104  1 !  11   25-APR-78       K.D. MORSE          CONVERT TO NATIVE COMPILER.
;   105          0105  1 !  12   12-MAY-78       K.D. MORSE          NO CHANGES FOR 50.
;   106          0106  1 !                                           INIT_MC_GLOBALS NOW SETS THE
;   107          0107  1 !                                           MC_IN_RST BIT TO FALSE FOR THE
;   108          0108  1 !                                           SPECIAL GLOBAL MC.  THIS IS
;   109          0109  1 !                                           NECESSARY BECAUSE THE SAT/LVT
;   110          0110  1 !                                           ACCESS FUNCTIONS MAY LOOK THERE
;   111          0111  1 !                                           BUT THERE ARE NO STORAGE
;   112          0112  1 !                                           DESCRIPTORS. (51)
;   113          0113  1 !  13   18-MAY-78       K.D. MORSE          NO CHANGES FOR VERS 52.
;   114          0114  1 !  14   18-MAY-78       K.D. MORSE          NO CHANGES FOR VERS 53.
```

```
;  115          0115  1 !  15   13-JUN-78        K.D. MORSE          ADD FAO COUNTS TO SIGNALS.
;  116          0116  1 !  16   28-JUN-78        K.D. MORSE          CHECK FOR DUPLICATE MODULE NAMES
;  117          0117  1 !                                           IN PAT$BUILD_MC AND IF FOUND
;  118          0118  1 !                                           THEN SKIP SECOND ONE THROUGH
;  119          0119  1 !                                           THE NEXT EOM RECORD. (54)
;  120          0120  1 !                                           NO CHANGES FOR 55-57.
;  121          0121  1 !
;  122          0122  1 !--
```

```
;  124          0123  1 !
;  125          0124  1 ! TABLE OF CONTENTS:
;  126          0125  1 !
;  127          0126  1
;  128          0127  1 FORWARD ROUTINE
;  129          0128  1         PAT$INIT_RST : NOVALUE,                 ! Do the RST initialization.
;  130          0129  1         INIT_MC_GLOBALS : NOVALUE,              ! Initialize the MC chain.
;  131          0130  1         BUILD_MC,                               ! Build the Module Chain (MC).
;  132          0131  1         BUILD_RST,                              ! Initialize the RST so that it
;  133          0132  1                                                 !  contains a given set of modules.
;  134          0133  1         OK_TO_ADD,                              ! Check, initialize, and set up to ADD_MODUL
;  135          0134  1         PAT$ADD_MODULE,                         ! Add a module to the RST.
;  136          0135  1         ADD_NT,                                 ! Add a DST symbol to the RST.
;  137          0136  1         ADD_SAT,                                ! Make a SAT entry for a symbol.
;  138          0137  1         ADD_LVT,                                ! Make an LVT entry for a symbol.
;  139          0138  1         PAT$GET_BOUNDS : NOVALUE;               ! Calculate the bounds of an array.
;  140          0139  1
;  141          0140  1 !
;  142          0141  1 ! INCLUDE FILES:
;  143          0142  1 !
;  144          0143  1
;  145          0144  1 LIBRARY 'SYS$LIBRARY:LIB.L32';
;  146          0145  1 REQUIRE 'SRC$:PATPCT.RFQ';                      ! Define PATCH PSECTS
;  147          0185  1 REQUIRE 'LIB$:PATDEF.REQ';                  ! Defines literals
;  148          0239  1 REQUIRE 'LIB$:PATMSG.REQ';
;  149          0413  1 REQUIRE 'SRC$:VXSMAC.REQ';
;  150          0478  1 REQUIRE 'SRC$:PATGEN.REQ';
;  151          0700  1 REQUIRE 'SRC$:PATRTS.REQ';
;  152          1796  1 REQUIRE 'SRC$:SYSSER.REQ';
```

```
;  R1828  1      SWITCHES LIST (SOURCE);
;  R1829  1
;  R1830  1      EXTERNAL ROUTINE
;  R1831  1              PAT$fao_out;            ! formats a line and outputs to the terminal
;  R1832  1
```

```
;  153          1878  1
;  154          1879  1 !
;  155          1880  1 ! MACROS:
;  156          1881  1 !
;  157          1882  1
;  158          1883  1 !
;  159          1884  1 ! EQUATED SYMBOLS:
;  160          1885  1 !
;  161          1886  1
;  162          1887  1
;  163          1888  1 !
;  164          1889  1 ! EXTERNAL REFERENCES:
;  165          1890  1 !
;  166          1891  1
;  167          1892  1 EXTERNAL
;  168          1893  1         PAT$GB_SYMBOLS : BYTE,                          ! Indicator if symbols are in the image
;  169          1894  1         PAT$GL_RST_BEGN,                                ! Starting address of RST
;  170          1895  1         PAT$GL_MC_PTR   : REF MC_RECORD,                ! Pointer to the Module Chain (MC).
;  171          1896  1         PAT$GL_NT_HASH  : REF RST_POINTER;              ! Pointer to the name table (NT) hash vector
;  172          1897  1
;  173          1898  1 EXTERNAL ROUTINE
;  174          1899  1         PAT$FIND_MODULE,                                ! Scans module chanin for a module name
;  175          1900  1         PAT$SORT_SA_LVT : NOVALUE,                      ! Sort the SAT or LVT
;  176          1901  1         PAT$VS_INIT : NOVALUE,                          ! Initialize a vector storage area.
;  177          1902  1         PAT$VS_GET,                                     ! Allocate records from a so-called
;  178          1903  1                                                         !  "vector storage" vector.
;  179          1904  1         PAT$VS_FREE : NOVALUE,                          ! Release vector storage.
;  180          1905  1         PAT$VS_SHRINK : NOVALUE,                        ! Free up unused vector storage.
;  181          1906  1         PAT$MODULE_SIZE,                                ! Deduce how much RST space adding
;  182          1907  1                                                         !   a module will take.
;  183          1908  1         PAT$DST_VALUE,                                  ! Evaluate a DST
;  184          1909  1         PAT$SYMBOL_VALU,                                ! Evaluate a symbol given its NT_PTR.
;  185          1910  1         PAT$NT_HASH_FCN,                                ! Hashing function for calculating
;  186          1911  1                                                         !   dispersal of NT entries.
;  187          1912  1         PAT$POSITON_DST,                                ! Make a certain DST record available
;  188          1913  1                                                         !   and 'SRC$:position' the DST for GET_NXT_
;  189          1914  1         PAT$GET_NXT_DST,                                ! Make the next DST record available.
;  190          1915  1         PAT$GET_NXT_GST,                                ! Make the next GST record available.
;  191          1916  1
;  192          1917  1         PAT$GET_NXT_SAT,                                ! Sequential access to the SAT.
;  193          1918  1         PAT$GET_NXT_LVT,                                ! Sequential access to the LVT.
;  194          1919  1
;  195          1920  1         !++
;  196          1921  1         ! We use two storage allocators - one when we want to allocate or
;  197          1922  1         ! free up storage we want to address via RST-pointers, and one when
;  198          1923  1         ! we deal with free storage in the same way that the rest
;  199          1924  1         ! of PATCH does.
;  200          1925  1         !--
;  201          1926  1
;  202          1927  1         PAT$FREEZ,                                      ! Allocate and zero a block of ordinary stor
;  203          1928  1         PAT$RST_FREEZ,                                  ! Allocate and zero RST-pointer storage.
;  204          1929  1         PAT$REPORT_FREE;                                ! Report on remaining available RST storage.
;  205          1930  1
;  206          1931  1 !
;  207          1932  1 ! OWN STORAGE:
;  208          1933  1 !
;  209          1934  1
```

```
;  210          1935  1 !++
;  211          1936  1 ! We maintain a pointer to the MC record for the module we are currently
;  212          1937  1 ! (de)allocating vector storage for.  This is done simply to avoid passing this
;  213          1938  1 ! parameter around.
;  214          1939  1 !--
;  215          1940  1
;  216          1941  1 OWN
;  217          1942  1         CURRENT_MODULE : REF MC_RECORD;
```

```
;  219     1943  1 GLOBAL ROUTINE PAT$INIT_RST : NOVALUE =
;  220     1944  1
;  221     1945  1 !++
;  222     1946  1 ! FUNCTIONAL DESCRIPTION:
;  223     1947  1 !
;  224     1948  1 !       Do the once-only initialization of PATCH's
;  225     1949  1 !       Runtime Symbol Table (RST) data structures.
;  226     1950  1 !
;  227     1951  1 ! FORMAL PARAMETERS:
;  228     1952  1 !
;  229     1953  1 !       none
;  230     1954  1 !
;  231     1955  1 ! IMPLICIT INPUTS:
;  232     1956  1 !
;  233     1957  1 !       In general, we expect that error reporting is done at
;  234     1958  1 !       the innermost level.  This means that we simply
;  235     1959  1 !       return bad status codes all the way out, without saying
;  236     1960  1 !       anything.
;  237     1961  1 !
;  238     1962  1 ! IMPLICIT OUTPUTS:
;  239     1963  1 !
;  240     1964  1 !       none
;  241     1965  1 !
;  242     1966  1 ! COMPLETION CODES:
;  243     1967  1 !
;  244     1968  1 !       NOVALUE - the only thing which can go wrong that we
;  245     1969  1 !                 don't handle is running out of free storage
;  246     1970  1 !                 when we ask for it blindly (i.e. not checking
;  247     1971  1 !                 that there is enough before we ask for it).
;  248     1972  1 !               - This must fail because we only assume that we
;  249     1973  1 !                 have enough storage for the NT hash vector and
;  250     1974  1 !                 a 1-element MC chain.  If there is really
;  251     1975  1 !                 not enough for this, PATCH is surely in trouble.
;  252     1976  1 !               - The storage manager doesn't return control to us
;  253     1977  1 !                 in any case...
;  254     1978  1 !
;  255     1979  1 ! SIDE EFFECTS:
;  256     1980  1 !
;  257     1981  1 !       The RST gets initialized.
;  258     1982  1 !
;  259     1983  1 !--
;  260     1984  1
;  261     1985  2 BEGIN
;  262     1986  2
;  263     1987  2 LOCAL
;  264     1988  2         LIST_ALL,
;  265     1989  2         PTR     : REF VECTOR[,BYTE];
;  266     1990  2
;  267     1991  2 !++
;  268     1992  2 ! Before we can do much we must have the NT hash vector.  It is a fixed size and
;  269     1993  2 ! gets allocated once-and-for-all.
;  270     1994  2 !--
;  271     1995  2 PAT$GL_NT_HASH = PAT$FREEZ( RST_UNITS(NT_HASH_SIZE * %SIZE(NT_HASH_RECORD)));
;  272     1996  2
;  273     1997  2 !++
;  274     1998  2 ! Add the global symbol definitions to the RST structures, and start off the
;  275     1999  2 ! module chain with the 'fake' MC record reserved for globals. This code assumes
```

```
;  276          2000  2 ! that PAT$GET_NXT_GST has already been set up so that the first time we call
;  277          2001  2 ! it the first GST record is returned.  Begin adding MC's to the chain in a
;  278          2002  2 ! forward-linked fashion.
;  279          2003  2 !--
;  280          2004  2 INIT_MC_GLOBALS();
;  281          2005  2
;  282          2006  2 !++
;  283          2007  2 ! Build the Module Chain, giving up if it doesn't work.  Note that this routine
;  284          2008  2 ! only works when the 'next' record, as defined by PAT$GET_NXT_DST, is the first
;  285          2009  2 ! record in the DST.  If we really wanted to do some manipulating of the DST
;  286          2010  2 ! before building the module chain, we would have to invent a fake REWIND_DST or
;  287          2011  2 ! something, or pass BUILD_MC the ID of where to start.  This would mean
;  288          2012  2 ! that BUILD_MC would have to 'special-case' the first record instead of
;  289          2013  2 ! just calling 'next' for them all.
;  290          2014  2 !--
;  291          2015  3 IF (NOT BUILD_MC())
;  292          2016  2 THEN
;  293          2017  2         RETURN;
;  294          2018  2
;  295          2019  2 !++
;  296          2020  2 ! Build an RST that contains those modules flagged as eligible in the module
;  297          2021  2 ! chain. We also pass on the estimate of the number of globals that the RST
;  298          2022  2 ! will contain.
;  299          2023  2 !--
;  300          2024  3 IF (NOT BUILD_RST())
;  301          2025  2 THEN
;  302          2026  2         RETURN;
;  303          2027  2
;  304          2028  2 !++
;  305          2029  2 ! Sort the SAT and LVT vectors so that we can access them (later) more efficiently.
;  306          2030  2 !--
;  307          2031  2 PAT$SORT_SA_LVT( PAT$GET_NXT_SAT );
;  308          2032  2 PAT$SORT_SA_LVT( PAT$GET_NXT_LVT );
;  309          2033  2
;  310          2034  1 END;


                                                        .TITLE  PATBLD
                                                        .IDENT  \V04-000\

                                                        .PSECT  _PAT$OWN,NOEXE,2

                                                00000 CURRENT_MODULE:
                                                        .BLKB   4

                                                        .EXTRN  PAT$FAO_OUT, PAT$GB_SYMBOLS
                                                        .EXTRN  PAT$GL_RST_BEGN
                                                        .EXTRN  PAT$GL_MC_PTR, PAT$GL_NT_HASH
                                                        .EXTRN  PAT$FIND_MODULE
                                                        .EXTRN  PAT$SORT_SA_LVT
                                                        .EXTRN  PAT$VS_INIT, PAT$VS_GET
                                                        .EXTRN  PAT$VS_FREE, PAT$VS_SHRINK
                                                        .EXTRN  PAT$MODULE_SIZE
                                                        .EXTRN  PAT$DST_VALUE, PAT$SYMBOL_VALU
                                                        .EXTRN  PAT$NT_HASH_FCN
                                                        .EXTRN  PAT$POSITON_DST
                                                        .EXTRN  PAT$GET_NXT_DST
```

```
                                        .EXTRN  PAT$GET_NXT_GST
                                        .EXTRN  PAT$GET_NXT_SAT
                                        .EXTRN  PAT$GET_NXT_LVT
                                        .EXTRN  PAT$FREEZ, PAT$RST_FREEZ
                                        .EXTRN  PAT$REPORT_FREE

                                        .PSECT  _PAT$CODE,NOWRT,2

                            0004 00000          .ENTRY  PAT$INIT_RST, Save R2                   ; 1943
                    52 00000000G    EF  9E 00002    MOVAB   PAT$SORT_SA_LVT, R2
                    7E          80  8F  9A 00009    MOVZBL  #128, -(SP)                         ; 1995
          00000000G EF              01  FB 0000D    CALLS   #1, PAT$FREEZ
          00000000G EF              50  D0 00014    MOVL    R0, PAT$GL_NT_HASH
          00000000V EF              00  FB 0001B    CALLS   #0, INIT_MC_GLOBALS                 ; 2004
          00000000V EF              00  FB 00022    CALLS   #0, BUILD_MC                        ; 2015
                    1C              50  E9 00029    BLBC    R0, 1$
          00000000V EF              00  FB 0002C    CALLS   #0, BUILD_RST                       ; 2024
                    12              50  E9 00033    BLBC    R0, 1$
                        00000000G   EF  9F 00036    PUSHAB  PAT$GET_NXT_SAT                     ; 2031
                    62              01  FB 0003C    CALLS   #1, PAT$SORT_SA_LVT
                        00000000G   EF  9F 0003F    PUSHAB  PAT$GET_NXT_LVT                     ; 2032
                    62              01  FB 00045    CALLS   #1, PAT$SORT_SA_LVT
                                    04  00048 1$:   RET                                         ; 2034

; Routine Size:  73 bytes,    Routine Base:  _PAT$CODE + 0000
```

```
;  312     2035  1 ROUTINE INIT_MC_GLOBALS . NOVALUE =
;  313     2036  1
;  314     2037  1 !++
;  315     2038  1 ! Functional Description:
;  316     2039  1 !
;  317     2040  1 !       This routine initializes the module chain.  It builds the dummy
;  318     2041  1 !       entry which is always the first link in the module chain.  All
;  319     2042  1 !       the global symbols are linked to the dummy module. This routine
;  320     2043  1 !       does that part of RST initialization which MUST
;  321     2044  1 !       work if PATCH is to survive.
;  322     2045  1 !
;  323     2046  1 ! Formal Parameters:
;  324     2047  1 !
;  325     2048  1 !       NONE
;  326     2049  1 !
;  327     2050  1 ! Implicit Inputs:
;  328     2051  1 !
;  329     2052  1 !       none.
;  330     2053  1 !
;  331     2054  1 ! Routine Value:
;  332     2055  1 !
;  333     2056  1 !       NONE
;  334     2057  1 !
;  335     2058  1 ! Side Effects:
;  336     2059  1 !
;  337     2060  1 !       PAT$GL_MC_PTR gets initialized.
;  338     2061  1 !
;  339     2062  1 !--
;  340     2063  1
;  341     2064  2 BEGIN
;  342     2065  2
;  343     2066  2 !++
;  344     2067  2 ! Build the single MC record which must always exist. This one does NOT
;  345     2068  2 ! correspond to a module, but rather we use it to 'hang' the globals off.  The
;  346     2069  2 ! building of this record must go OK - the storage manager doesn't give us
;  347     2070  2 ! back control.
;  348     2071  2 !--
;  349     2072  2 PAT$GL_MC_PTR = PAT$RST_FREEZ(RST_UNITS(RST_MC_SIZE));
;  350     2073  2
;  351     2074  2 !++
;  352     2075  2 ! Initialize the few fields which this special MC record must have.  We assume
;  353     2076  2 ! the rest of the record is 0.  The global MC is never marked 'MC_IN_RST'
;  354     2077  2 ! because globals are not really 'in' the RST.
;  355     2078  2 !--
;  356     2079  2 PAT$GL_MC_PTR [MC_IS_GLOBAL] = TRUE;
;  357     2080  2 PAT$GL_MC_PTR [MC_IN_RST] = FALSE;
;  358     2081  2 PAT$GL_MC_PTR [MC_TYPE] = DSC$K_DTYPE_MOD;
;  359     2082  2
;  360     2083  2 !++
;  361     2084  2 ! Permanently allocate the storage which 'hangs' off this MC entry.  Since
;  362     2085  2 ! the globals now remain in their own data base (the GST), the RST needs only
;  363     2086  2 ! one NT and one SAT record for 'cache' use - i.e., we fill them when we
;  364     2087  2 ! bring a GST entry into the RST for SYM_TO_VAL, etc.
;  365     2088  2 !--
;  366     2089  2 PAT$GL_MC_PTR[MC_GBL_NT_PTR] = PAT$RST_FREEZ(RST_UNITS(RST_NT_SIZE));
;  367     2090  2 PAT$GL_MC_PTR[MC_GBL_SAT_PTR] = PAT$FREEZ(RST_UNITS(RST_SAT_SIZE));
;  368     2091  2
```

```
;  369          2092  2 !++
;  370          2093  2 ! Sometimes we do not want the GST consulted in a SYM_TO_VAL, and some
;  371          2094  2 ! times we have consulted it and do not want some other call to overwrite the
;  372          2095  2 ! two cache GBL/RST records.  For this reason, the following field is turned
;  373          2096  2 ! on and off to indicate if the GST is 'locked' or not.
;  374          2097  2 !--
;  375          2098  2 PAT$GL_MC_PTR[MC_GBL_LOCKED] = FALSE;
;  376          2099  2 RETURN;
;  377          2100  1 END;


                                003C 00000 INIT_MC_GLOBALS:
                                           .WORD   Save R2,R3,R4,R5                                    ; 2035
                     55 00000000G   EF  9E 00002           MOVAB   PAT$RST_FREEZ, R5
                     54 00000000G   EF  9E 00009           MOVAB   PAT$GL_RST_BEGN, R4
                     53 00000000G   EF  9E 00010           MOVAB   PAT$GL_MC_PTR, R3
                                    0F  DD 00017           PUSHL   #15                                 ; 2072
                     65             01  FB 00019           CALLS   #1, PAT$RST_FREEZ
                     63             50  D0 0001C           MOVL    R0, PAT$GL_MC_PTR
                52   63             64  C1 0001F           ADDL3   PAT$GL_RST_BEGN, PAT$GL_MC_PTR, R2  ; 2079
                   03  A2           01  88 00023           BISB2   #1, 3(R2)
                   03  A2           02  8A 00027           BICB2   #2, 3(R2)                           ; 2080
                   02  A2     BC    8F  90 0002B           MOVB    #-68, 2(R2)                         ; 2081
                                    0B  DD 00030           PUSHL   #11                                 ; 2089
                     65             01  FB 00032           CALLS   #1, PAT$RST_FREEZ
                   04  A2           50  B0 00035           MOVW    R0, 4(R2)
                52   63             64  C1 00039           ADDL3   PAT$GL_RST_BEGN, PAT$GL_MC_PTR, R2  ; 2090
                                    03  DD 0003D           PUSHL   #3
                00000000G   EF      01  FB 0003F           CALLS   #1, PAT$FREEZ
                     35  A2         50  D0 00046           MOVL    R0, 53(R2)
                50   63             64  C1 0004A           ADDL3   PAT$GL_RST_BEGN, PAT$GL_MC_PTR, R0  ; 2098
                   03  A0           04  8A 0004E           BICB2   #4, 3(R0)
                                        04 00052           RET                                         ; 2100

; Routine Size:  83 bytes,    Routine Base:  _PAT$CODE + 0049
```

```
;  379     2101  1 ROUTINE BUILD_MC =
;  380     2102  1
;  381     2103  1 !++
;  382     2104  1 ! FUNCTIONAL DESCRIPTION:
;  383     2105  1 !
;  384     2106  1 !       This routine builds the Module Chain (MC) directly from the DST.
;  385     2107  1 !       This is part of a once-only PATCH initialization procedure.
;  386     2108  1 !
;  387     2109  1 !       If there is no DST (because none of the module
;  388     2110  1 !       were compiled with /debug symbols on), then this
;  389     2111  1 !       routine simply returns(true) when the first attempt
;  390     2112  1 !       it makes to fetch a DST record returns EOF.
;  391     2113  1 !
;  392     2114  1 ! FORMAL PARAMETERS:
;  393     2115  1 !
;  394     2116  1 !       none
;  395     2117  1 !
;  396     2118  1 ! IMPLICIT INPUTS:
;  397     2119  1 !
;  398     2120  1 !       -It is assumed that the first call to PAT$GET_NXT_DST
;  399     2121  1 !       that we make will retreive us the first record
;  400     2122  1 !       in the DST.
;  401     2123  1 !
;  402     2124  1 !       -The free-storage manager zeros-out the storage returned.
;  403     2125  1 !       (We therefore don't bother to zero-out
;  404     2126  1 !       link fields, etc, in newly-obtained MC records).
;  405     2127  1 !
;  406     2128  1 !       -We assume that PAT$GL_MC_PTR has already been set to
;  407     2129  1 !        point to the 'dummy' global MC record.
;  408     2130  1 !
;  409     2131  1 ! IMPLICIT OUTPUTS:
;  410     2132  1 !
;  411     2133  1 !       The MC is built - it contains a host of implications.
;  412     2134  1 !
;  413     2135  1 ! ROUTINE VALUE:
;  414     2136  1 !
;  415     2137  1 !       NONE
;  416     2138  1 !
;  417     2139  1 ! COMPLETION CODES:
;  418     2140  1 !
;  419     2141  1 !       TRUE, if all goes OK,
;  420     2142  1 !       FALSE, otherwise.
;  421     2143  1 !
;  422     2144  1 ! SIDE EFFECTS:
;  423     2145  1 !
;  424     2146  1 !       -The MC is built.
;  425     2147  1 !       -The DST is 'read' straight through to the end unless going any further
;  426     2148  1 !        would take too much of the RST just for MC's.  In this case, the last
;  427     2149  1 !        part of the DST is henceforth ignored by PATCH.
;  428     2150  1 !
;  429     2151  1 !--
;  430     2152  1
;  431     2153  2 BEGIN
;  432     2154  2
;  433     2155  2 LOCAL
;  434     2156  2         SAT_COUNT : VOLATILE,                   ! Count up per-module SAT entries,
;  435     2157  2         LVT_COUNT : VOLATILE,                   ! Count up per-module LVT entries,
```

```
;  436          2158  2          NT_COUNT : VOLATILE,                                ! Count up per-module NT entries,
;  437          2159  2          IN_MODULE,                                          ! Flag to say whether we are 'in between'
;  438          2160  2                                                              !  MODULE_BEGIN - MODULE_END pairs.
;  439          2161  2          DST_REC_ID,                                         ! Record ID passed back by PAT$GET_NXT_DST.
;  440          2162  2          CURRENT : REF MC_RECORD,                            ! Pointer to the MC record we are
;  441          2163  2                                                              ! currently initializing.
;  442          2164  2          DST_RECRD : REF DST_RECORD;                         ! Pointer to the current record
;  443          2165  2                                                              ! of the DST we are reading.
;  444          2166  2
;  445          2167  2  !++
;  446          2168  2  ! Initialize the context flag IN_MODULE, and begin going thru the DST
;  447          2169  2  ! sequentially. We assume that someone else, namely PATCH's INIT routine, has
;  448          2170  2  ! made sure that the DST is available and that the 'next' record is the first
;  449          2171  2  ! one.  The chain starts off with the 'fake' global MC record.  Other MC's are
;  450          2172  2  ! added in forward-linked fashion.
;  451          2173  2  !--
;  452          2174  2  CURRENT = .PAT$GL_MC_PTR;
;  453          2175  2  IN_MODULE = FALSE;
;  454          2176  3  WHILE( (DST_RECRD = PAT$GET_NXT_DST( DST_REC_ID )) NEQ 0 )
;  455          2177  2  DO
;  456          2178  3          BEGIN
;  457          2179  3          !++
;  458          2180  3          ! Process each record depending on its DST type.
;  459          2181  3          !--
;  460          2182  3          CASE .DST_RECRD [DSTR_TYPE] FROM DST_DST_LOWEST TO DST_DST_HIGHEST OF
;  461          2183  3
;  462          2184  3              SET
;  463          2185  3
;  464          2186  3              [DSC$K_DTYPE_MOD]:                              ! Module Begin Record.
;  465          2187  4                  BEGIN
;  466          2188  4                  LOCAL
;  467          2189  4                          MOD_NAME_DESC : BLOCK[8,BYTE],
;  468          2190  4                          NEW_PTR : REF MC_RECORD;
;  469          2191  5                  IF (.IN_MODULE)                             ! Check for misplaced MODULE record.
;  470          2192  4                  THEN
;  471          2193  5                          BEGIN
;  472          2194  5                          $FAO_TT_OUT( 'Module Begin Phase Error.' );
;  473          2195  5                          RETURN(FALSE);
;  474          2196  4                          END;
;  475          2197  4
;  476          2198  4                  !++
;  477          2199  4                  ! Now check to see if this module is already in the chain.
;  478          2200  4                  ! If found, skip until the next EOM record.
;  479          2201  4                  !--
;  480          2202  4                  MOD_NAME_DESC[DSC$W_LENGTH] = .DST_RECRD[DSTR_NAME];
;  481          2203  4                  MOD_NAME_DESC[DSC$A_POINTER] = DST_RECRD[DSTR_NAME] + 1;
;  482          2204  5                  IF (PAT$FIND_MODULE(MOD_NAME_DESC, FALSE) NEQ 0)
;  483          2205  4                  THEN
;  484          2206  5                          BEGIN
;  485        P 2207  5                          $FAO_TT_OUT('Duplicate module !AD ignored.',
;  486          2208  5                                  .MOD_NAME_DESC[DSC$W_LENGTH],.MOD_NAME_DESC[DSC$A_POINTER]);
;  487          2209  5                          DO
;  488          2210  5                          DST_RECRD = PAT$GET_NXT_DST(DST_REC_ID)
;  489          2211  5                          UNTIL .DST_RECRD[DSTR_TYPE] EQL DSC$K_DTYPE_EOM;
;  490          2212  5                          END
;  491          2213  4                  ELSE
;  492          2214  5                  BEGIN
```

```
;  493          2215  5              !++
;  494          2216  5              ! See if taking the RST space for this new MC record still
;  495          2217  5              ! leaves a reasonable amount for subsequent "SET MODULE"
;  496          2218  5              ! commands, (i.e., 50% of the RST).  If this is not the case,
;  497          2219  5              ! then ignore the rest of the modules and exit from this loop.
;  498          2220  5              !--
;  499          2221  5              IF NOT PAT$REPORT_FREE() GTR RST_MODU_SIZE
;  500          2222  5              THEN
;  501          2223  6                      BEGIN
;  502          2224  6                      !++
;  503          2225  6                      ! Report informational message.  This routine must
;  504          2226  6                      ! do a normal return and not an UNWIND.
;  505          2227  6                      !--
;  506          2228  6                      SIGNAL(PAT$_LONGDST);
;  507          2229  6                      EXITLOOP;
;  508          2230  5                      END;
;  509          2231  5
;  510          2232  5              !++
;  511          2233  5              ! Allocate space for the new MC record.
;  512          2234  5              !--
;  513          2235  6              IF ((NEW_PTR = PAT$RST_FREEZ(RST_UNITS(RST_MC_SIZE))) EQL 0)
;  514          2236  5              THEN
;  515          2237  5                      RETURN(FALSE);
;  516          2238  5              CURRENT [MC_NEXT] = .NEW_PTR;                   ! Link this one onto the previous one.
;  517          2239  5
;  518          2240  5              !++
;  519          2241  5              ! Initialize the new 'current' node, and those fields belonging
;  520          2242  5              ! to it that we know the value of at this point. (Some values
;  521          2243  5              ! get initialized only after  we have read the DST for this
;  522          2244  5              ! module).
;  523          2245  5              !--
;  524          2246  5              IN_MODULE = TRUE;
;  525          2247  5              CURRENT = .NEW_PTR;
;  526          2248  5              SAT_COUNT = 0;
;  527          2249  5              LVT_COUNT = 0;
;  528          2250  5              NT_COUNT = 0;
;  529          2251  5              CURRENT [MC_TYPE] = .DST_RECRD [DSTR_TYPE];
;  530          2252  5              CURRENT [MC_DST_START] = .DST_REC_ID;
;  531          2253  5              CURRENT [MC_IS_MAIN] = FALSE;
;  532          2254  5              CURRENT [MC_IS_GLOBAL] = FALSE;
;  533          2255  5
;  534          2256  5              !++
;  535          2257  5              ! Encode the language indicator into 3 bits. This only
;  536          2258  5              ! accomodates values 0-7, enough for now.  We may have to
;  537          2259  5              ! enlarge this field later.
;  538          2260  5              !--
;  539          2261  5              CURRENT [MC_LANGUAGE] = .DST_RECRD [DSTR_VALUE];
;  540          2262  5
;  541          2263  5              !++
;  542          2264  5              ! If we want all modules to be IN the RST by default, set the
;  543          2265  5              ! following to TRUE.  Otherwise set it to FALSE, and make the
;  544          2266  5              ! decision as to whether or not we want a module after its
;  545          2267  5              ! DST has been read.  (See DSC$K_DTYPE_EOM, below).
;  546          2268  5              !--
;  547          2269  5              CURRENT [MC_IN_RST] = FALSE;
;  548          2270  5
;  549          2271  5              !++
```

```
;  550          2272  5                ! Moving the name could cause problems if we change the data
;  551          2273  5                ! representation.  Note that here we move both the name-size
;  552          2274  5                ! count as well as the name itself.
;  553          2275  5                !--
;  554          2276  5                CH$MOVE( .DST_RECRD [DSTR_NAME] + 1,
;  555          2277  5                         DST_RECRD [DSTR_NAME],
;  556          2278  5                         CURRENT [MC_NAME_CS]);
;  557          2279  4                END;
;  558          2280  3                END;
;  559          2281  3
;  560          2282  3            [DSC$K_DTYPE_EOM]:                              ! Module End Record.
;  561          2283  4                BEGIN
;  562          2284  5                IF (NOT .IN_MODULE)                         . Check for misplaced END_MODULE record.
;  563          2285  4                THEN
;  564          2286  5                        BEGIN
;  565          2287  5                        $FAO_TT_OUT( '!/Module End Phase Error.' );
;  566          2288  5                        RETURN(FALSE);
;  567          2289  4                        END;
;  568          2290  4
;  569          2291  4                !++
;  570          2292  4                ! Reset the 'in MODULE' context flag, and initialize the
;  571          2293  4                ! settings of all MC flag bits.
;  572          2294  4                !--
;  573          2295  4                IN_MODULE = FALSE;
;  574          2296  4
;  575          2297  4                !++
;  576          2298  4                ! Record the module statistics.
;  577          2299  4                !--
;  578          2300  4                CURRENT [MC_STATICS] = .SAT_COUNT;
;  579          2301  4                CURRENT [MC_LITERALS] = .LVT_COUNT;
;  580          2302  4                CURRENT [MC_NAMES] = .NT_COUNT;
;  581          2303  3                END;
;  582          2304  3
;  583          2305  3            [DSC$K_DTYPE_RTN,                               ! Routine DSTs.
;  584          2306  3             DSC$K_DTYPE_SLB]:                              ! Labels in FORTRAN and BLISS.
;  585          2307  4                BEGIN
;  586          2308  4                !++
;  587          2309  4                ! Just tally up the needed statistics so that we can build
;  588          2310  4                ! the other data  structures later.
;  589          2311  4                !--
;  590          2312  4                NT_COUNT = .NT_COUNT +1;
;  591          2313  4                SAT_COUNT = .SAT_COUNT +1;
;  592          2314  3                END;
;  593          2315  3
;  594          2316  3            [DSC$K_DTYPE_EOR,                               ! BLISS-only End-of-Routine.
;  595          2317  3             DSC$K_DTYPE_FLD]:                              ! BLISS-only FIELD records.
;  596          2318  3                ;                                           ! We can safely ignore these for now.
;  597          2319  3
;  598          2320  3            [DSC$K_DTYPE_LBL]:                              ! Label or Literal DSTs. (MARS only)
;  599          2321  4                BEGIN
;  600          2322  4                !++
;  601          2323  4                ! This contributes 1 to NT, and another one to either LVT or SAT.
;  602          2324  4                !--
;  603          2325  4                NT_COUNT = .NT_COUNT +1;
;  604          2326  5                IF (.DST_RECRD [DSTR_ACC01] EQL ACCS_VALUEADR)
;  605          2327  4                THEN
;  606          2328  4                        SAT_COUNT = .SAT_COUNT + 1          ! The symbol is an ADDRESS.
```

```
;  607     2329  4                ELSE
;  608     2330  5                        IF (.DST_RECRD [DSTR_ACC01] EQL ACCS_LITERAL)
;  609     2331  4                        THEN
;  610     2332  4                                LVT_COUNT = .LVT_COUNT +1;      ! The symbol is a LITERAL.
;  611     2333  3                END;
;  612     2334  3
;  613     2335  3            [DSC$K_DTYPE_PCT]:                                  ! PSECT DSTs.
;  614     2336  4                BEGIN
;  615     2337  5                IF (NOT.IN_MODULE )
;  616     2338  4                THEN
;  617     2339  5                        BEGIN
;  618     2340  5                        $FAO_TT_OUT('!/Misplaced PSECT.');
;  619     2341  5                        RETURN(FALSE);
;  620     2342  4                        END;
;  621     2343  4                NT_COUNT = .NT_COUNT +1;                        ! PSECTs also count as a symbol name.
;  622     2344  4                SAT_COUNT = .SAT_COUNT +1;
;  623     2345  3                END;
;  624     2346  3
;  625     2347  3            [INRANGE, OUTRANGE]:
;  626     2348  4                BEGIN
;  627     2349  4                !++
;  628     2350  4                ! The only reason for not making the "SRM types"
;  629     2351  4                ! part of the above CASE is because of the huge
;  630     2352  4                ! case table which gets generated otherwise.
;  631     2353  4                !--
;  632     2354  5                IF (.DST_RECRD [DSTR_TYPE] EQL DSC$K_DTYPE_Z)
;  633     2355  4                THEN
;  634     2356  5                        BEGIN
;  635     2357  5                        !++
;  636     2358  5                        ! BLISS type ZERO records. Whatever symbol this is,
;  637     2359  5                        ! it contributes a name, for sure, and either a literal
;  638     2360  5                        ! or a static.  We assume the worst!
;  639     2361  5                        !--
;  640     2362  5                        NT_COUNT = .NT_COUNT +1;
;  641     2363  5                        LVT_COUNT = .LVT_COUNT +1;
;  642     2364  5                        SAT_COUNT= .SAT_COUNT +1;
;  643     2365  5                        END
;  644     2366  4                ELSE
;  645     2367  5                IF (.DST_RECRD [DSTR_TYPE] LEQ DST_TYP_HIGHEST)
;  646     2368  4                THEN
;  647     2369  5                        BEGIN
;  648     2370  5                        !++
;  649     2371  5                        ! These types are candidates for the LVT and NT tables only.
;  650     2372  5                        !--
;  651     2373  5                        NT_COUNT = .NT_COUNT +1;
;  652     2374  5                        SAT_COUNT = .SAT_COUNT +1;
;  653     2375  5                        END
;  654     2376  4                ELSE
;  655     2377  5                        BEGIN
;  656     2378  5                        !++
;  657     2379  5                        ! Probably an error in the DST data.
;  658     2380  5                        !--
;  659     2381  5                        TRUE;
;  660     2382  4                        END;
;  661     2383  3                END;
;  662     2384  3            TES;
;  663     2385  2        END;                                                    ! Go back and process the next DST record.
```

```
;  664          2386  2 !++
;  665          2387  2 ! Make sure we didn't run off the end of the DST unexpectedly.  This means that
;  666          2388  2 ! we insist that EOM be the last record encountered.
;  667          2389  2 !--
;  668          2390  3 IF (.IN_MODULE)
;  669          2391  2 THEN
;  670          2392  3         BEGIN
;  671          2393  3         $FAO_TT_OUT('Premature End-of-DST Encountered.' );
;  672          2394  3         RETURN(FALSE);
;  673          2395  2         END;
;  674          2396  2
;  675          2397  2 RETURN( TRUE );                                        ! The chain has been built OK.
;  676          2398  1 END;
; INFO#212                L1:2194
; Null expression appears in value-required context
; INFO#212                L1:2287
; Null expression appears in value-required context
; INFO#212                L1:2340
; Null expression appears in value-required context
; INFO#212                L1:2393
; Null expression appears in value-required context
```

```
                                                        .PSECT  _PAT$PLIT,NOWRT,NOEXE,0

                                                    19  00000 P.AAA:  .BYTE   25
68 50 20 6E 69 67 65 42 20 65 6C 75 64 6F 4D        00001         .ASCII  \Module Begin Phase Error.\
                     2E 72 6F 72 72 45 20 65 73     00010
                                                    1D  0001A P.AAB:  .BYTE   29
6C 75 64 6F 6D 20 65 74 61 63 69 6C 70 75 44        0001B         .ASCII  \Duplicate module !AD ignored.\
   2E 64 65 72 6F 6E 67 69 20 44 41 21 20 65        0002A
                                                    19  00038 P.AAC:  .BYTE   25
68 50 20 64 6E 45 20 65 6C 75 64 6F 4D 2F 21        00039         .ASCII  \!/Module End Phase Error.\
                     2E 72 6F 72 72 45 20 65 73     00048
                                                    12  00052 P.AAD:  .BYTE   18
45 53 50 20 64 65 63 61 6C 70 73 69 4D 2F 21        00053         .ASCII  \!/Misplaced PSECT.\
                                       2E 54 43     00062
                                                    21  00065 P.AAE:  .BYTE   33
6F 2D 64 6E 45 20 65 72 75 74 61 6D 65 72 50        00066         .ASCII  \Premature End-of-DST Encountered.\
72 65 74 6E 75 6F 63 6E 45 20 54 53 44 2D 66        00075
                                       2E 64 65     00084


                                                        .PSECT  _PAT$CODE,NOWRT,2

                                                OFFC 00000 BUILD_MC:
                                                        .WORD   Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11            ; 2101
                                5B 00000000G    EF  9E 00002         MOVAB   PAT$GET_NXT_DST, R11
                                5A 00000000G    EF  9E 00009         MOVAB   PAT$GL_RST_BEGN, R10
                                59 00000000'    EF  9E 00010         MOVAB   P.AAA, R9
                                5E              18  C2 00017         SUBL2   #24, SP
                                56 00000000G    EF  D0 0001A         MOVL    PAT$GL_MC_PTR, CURRENT          ; 2174
                                                58  D4 00021         CLRL    IN_MODULE                       ; 2175
                                                5E  DD 00023 1$:      PUSHL   SP                              ; 2176
                                6B              01  FB 00025         CALLS   #1, PAT$GET_NXT_DST             ;
```

```
                                57              50  D0 00028          MOVL    R0, DST_RECRD                                   ;
                                                03  12 0002B          BNEQ    2$                                              ;
                                            012F    31 0002D          BRW     20$                                             ;
                          08        B7  8F        01  A7  8F 00030 2$:  CASEB   1(DST_RECRD), #183, #8                        ; 2182
                0100            0012        0116    FFED     00036 3$:  .WORD   1$-3$,-                                         ;
                0120            00DF        0029    0120     0003E              16$-3$,-                                        ;
                                                    FFED     00046              4$-3$,-                                         ;
                                                                                15$-3$,-                                        ;
                                                                                17$-3$,-                                        ;
                                                                                6$-3$,-                                         ;
                                                                                13$-3$,-                                        ;
                                                                                17$-3$,-                                        ;
                                                                                1$-3$                                           ;
                                                01  A7  95 00048 4$:  TSTB    1(DST_RECRD)                                    ; 2354
                                                09  12 0004B          BNEQ    5$                                              ;
                                            0C  AE  D6 0004D          INCL    NT_COUNT                                        ; 2362
                                            10  AE  D6 00050          INCL    LVT_COUNT                                       ; 2363
                                                0103    31 00053      BRW     18$                                             ; 2364
                                17              01  A7  91 00056 5$:  CMPB    1(DST_RECRD), #23                               ; 2367
                                                    C7  1A 0005A      BGTRU   1$                                              ;
                                                00F7    31 0005C      BRW     17$                                             ; 2373
                                07              58  E9 0005F 6$:  BLBC    IN_MODULE, 7$                                   ; 2191
                                                7E  D4 00062          CLRL    -(SP)                                           ; 2194
                                                59  DD 00064          PUSHL   R9                                              ;
                                                00FE    31 00066      BRW     21$                                             ;
                          04    AE          07  A7  9B 00069 7$:  MOVZBW  7(DST_RECRD), MOD_NAME_DESC                      ; 2202
                          08    AE          08  A7  9E 0006E          MOVAB   8(R7), MOD_NAME_DESC+4                          ; 2203
                                                7E  D4 00073          CLRL    -(SP)                                           ; 2204
                                            08  AE  9F 00075          PUSHAB  MOD_NAME_DESC                                   ;
                00000000G       EF              02  FB 00078          CALLS   #2, PAT$FIND_MODULE                             ;
                                                50  D5 0007F          TSTL    R0                                              ;
                                                22  13 00081          BEQL    9$                                              ;
                                            08  AE  DD 00083          PUSHL   MOD_NAME_DESC+4                                 ; 2208
                                7E          08  AE  3C 00086          MOVZWL  MOD_NAME_DESC, -(SP)                            ;
                                            1A  A9  9F 0008A          PUSHAB  P.AAB                                           ;
                00000000G       EF              03  FB 0008D          CALLS   #3, PAT$FAO_OUT                                 ;
                                                5E  DD 00094 8$:  PUSHL   SP                                              ; 2210
                                6B              01  FB 00096          CALLS   #1, PAT$GET_NXT_DST                             ;
                                57              50  D0 00099          MOVL    R0, DST_RECRD                                   ;
                          BD    8F          01  A7  91 0009C          CMPB    1(DST_RECRD), #189                              ; 2211
                                                F1  12 000A1          BNEQ    8$                                              ;
                                                6E  11 000A3          BRB     12$                                             ; 2204
                00000000G       EF              00  FB 000A5 9$:  CALLS   #0, PAT$REPORT_FREE                             ; 2221
                00007918        8F              50  D1 000AC          CMPL    R0, #31000                                      ;
                                                10  14 000B3          BGTR    10$                                             ;
                                    006D8003    8F  DD 000B5          PUSHL   #7176195                                        ; 2228
                00000000G       00              01  FB 000BB          CALLS   #1, LIB$SIGNAL                                  ;
                                            009A    31 000C2          BRW     20$                                             ; 2223
                                                0F  DD 000C5 10$:  PUSHL   #15                                             ; 2235
                00000000G       EF              01  FB 000C7          CALLS   #1, PAT$RST_FREEZ                               ;
                                                50  D5 000CE          TSTL    NEW_PTR                                         ;
                                                03  12 000D0          BNEQ    11$                                             ;
                                            009F    31 000D2          BRW     23$                                             ;
        51                      56              6A  C1 000D5 11$:  ADDL3   PAT$GL_RST_BEGN, CURRENT, R1                    ; 2238
                                61              50  B0 000D9          MOVW    NEW_PTR, (R1)                                   ;
                                58              01  D0 000DC          MOVL    #1, IN_MODULE                                   ; 2246
                                56              50  D0 000DF          MOVL    NEW_PTR, CURRENT                                ; 2247
```

```
                                                14  AE  D4 000E2          CLRL    SAT_COUNT                                       ; 2248
                                                10  AE  D4 000E5          CLRL    LVT_COUNT                                       ; 2249
                                                0C  AE  D4 000E8          CLRL    NT_COUNT                                        ; 2250
                        50              56              6A  C1 000EB          ADDL3   PAT$GL_RST_BEGN, CURRENT, R0                    ; 2251
                                02      A0      01      A7  90 000EF          MOVB    1(DST_RECRD), 2(R0)                             ;
                                04      A0              6E  D0 000F4          MOVL    DST_REC_ID, 4(R0)                               ; 2252
                                03      A0              05  8A 000F8          BICB2   #5, 3(R0)                                       ; 2254
        03      A0              03      03      03      A7  F0 000FC          INSV    3(DST_RECRD), #3, #3, 3(R0)                     ; 2261
                                03      A0              02  8A 00103          BICB2   #2, 3(R0)                                       ; 2269
                                        51      07      A7  9A 00107          MOVZBL  7(DST_RECRD), R1                                ; 2276
                                                        51  D6 0010B          INCL    R1                                              ;
                        0C      A0      07      A7          51  28 0010D          MOVC3   R1, 7(DST_RECRD), 12(R0)                ; 2278
                                                            47  11 00113 12$:      BRB     19$                                     ; 2182
                                        07                  58  E8 00115 13$:      BLBS    IN_MODULE, 14$                          ; 2284
                                                            7E  D4 00118          CLRL    -(SP)                                   ; 2287
                                                    38      A9  9F 0011A          PUSHAB  P.AAC                                   ;
                                                            48  11 0011D          BRB     21$                                     ;
                                                            58  D4 0011F 14$:      CLRL    IN_MODULE                               ; 2295
                        50              56                  6A  C1 00121          ADDL3   PAT$GL_RST_BEGN, CURRENT, R0            ; 2300
                                31      A0      14      AE  D0 00125          MOVL    SAT_COUNT, 49(R0)                       ;
                                35      A0      10      AE  D0 0012A          MOVL    LVT_COUNT, 53(R0)                       ; 2301
                                08      A0      0C      AE  D0 0012F          MOVL    NT_COUNT, 8(R0)                         ; 2302
                                                        26  11 00134          BRB     19$                                     ; 2182
                                                    0C      AE  D6 00136 15$:      INCL    NT_COUNT                                ; 2325
        01              02      A7              02          00  ED 00139          CMPZV   #0, #2, 2(DST_RECRD), #1                ; 2326
                                                            18  13 0013F          BEQL    18$                                     ;
                                        03              02  A7  93 00141          BITB    2(DST_RECRD), #3                        ; 2330
                                                            15  12 00145          BNEQ    19$                                     ;
                                                    10      AE  D6 00147          INCL    LVT_COUNT                               ; 2332
                                                            10  11 0014A          BRB     19$                                     ; 2182
                                        07                  58  E8 0014C 16$:      BLBS    IN_MODULE, 17$                          ; 2337
                                                            7E  D4 0014F          CLRL    -(SP)                                   ; 2340
                                                    52      A9  9F 00151          PUSHAB  P.AAD                                   ;
                                                            11  11 00154          BRB     21$                                     ;
                                                    0C      AE  D6 00156 17$:      INCL    NT_COUNT                                ; 2343
                                                    14      AE  D6 00159 18$:      INCL    SAT_COUNT                               ; 2344
                                                FEC4            31  0015C 19$:      BRW     1$                                      ; 2176
                                0E                          58  E9 0015F 20$:      BLBC    IN_MODULE, 22$                          ; 2390
                                                            7E  D4 00162          CLRL    -(SP)                                   ; 2393
                                                    65      A9  9F 00164          PUSHAB  P.AAE                                   ;
                00000000G       EF                          02  FB 00167 21$:      CALLS   #2, PAT$FAO_OUT                         ;
                                                            04  11 0016E          BRB     23$                                     ; 2394
                                50                          01  D0 00170 22$:      MOVL    #1, R0                                  ; 2397
                                                            04  00173          RET                                             ;
                                                            50  D4 00174 23$:      CLRL    R0                                      ; 2398
                                                            04  00176          RET                                             ;

; Routine Size:  375 bytes,    Routine Base:  _PAT$CODE + 009C
```

```
;  678     2399  1 ROUTINE BUILD_RST =
;  679     2400  1
;  680     2401  1 !++
;  681     2402  1 ! FUNCTIONAL DESCRIPTION:
;  682     2403  1 !
;  683     2404  1 !       Initialize the RST given a module chain.
;  684     2405  1 !
;  685     2406  1 ! FORMAL PARAMETERS:
;  686     2407  1 !
;  687     2408  1 !       none.
;  688     2409  1 !
;  689     2410  1 ! IMPLICIT INPUTS:
;  690     2411  1 !
;  691     2412  1 !       PAT$GL_MC_PTR   -is assumed to RST-point (in the sense
;  692     2413  1 !                        that an RST-pointer points) to the beginning
;  693     2414  1 !                        of the MC.  This MC is assumed to contain
;  694     2415  1 !                        all necessary information about those
;  695     2416  1 !                        modules marked elegible to be in the RST.
;  696     2417  1 !
;  697     2418  1 ! IMPLICIT OUTPUTS:
;  698     2419  1 !
;  699     2420  1 !       There will be at least RST_AVAIL_SIZE bytes of
;  700     2421  1 !       free storage left after this routine finishes.
;  701     2422  1 !
;  702     2423  1 ! ROUTINE VALUE:
;  703     2424  1 !
;  704     2425  1 !       none
;  705     2426  1 !
;  706     2427  1 ! COMPLETION CODES:
;  707     2428  1 !
;  708     2429  1 !       TRUE, if all went OK, FALSE otherwise.
;  709     2430  1 !
;  710     2431  1 ! SIDE EFFECTS:
;  711     2432  1 !
;  712     2433  1 !       A warning message is produced if any of the
;  713     2434  1 !       modules marked MC_IN_RST are not added to
;  714     2435  1 !       the RST.  This happens when we don't have
;  715     2436  1 !       enough space.
;  716     2437  1 !
;  717     2438  1 !--
;  718     2439  1
;  719     2440  2 BEGIN
;  720     2441  2
;  721     2442  2 LOCAL
;  722     2443  2         NOT_ALL_ADDED,                                  ! A flag to tell if we added all the
;  723     2444  2                                                         !  modules we wanted to.
;  724     2445  2         MC_PTR : REF MC_RECORD;                         ! Pointer to current MC record.
;  725     2446  2
;  726     2447  2 !++
;  727     2448  2 ! Initialize the flag that we use to detect when we didn't get to add all
;  728     2449  2 ! modules we wanted to.
;  729     2450  2 !--
;  730     2451  2 NOT_ALL_ADDED = FALSE;
;  731     2452  2
;  732     2453  2 !++
;  733     2454  2 ! Loop thru the module chain trying to add the indicated modules to the RST.
;  734     2455  2 !  Note that we purposely don't look at the first MC record in the chain because
```

```
;  735          2456  2 ! is is reserved for globals.
;  736          2457  2 !--
;  737          2458  2 MC_PTR = .PAT$GL_MC_PTR;
;  738          2459  3 WHILE( (MC_PTR = .MC_PTR [MC_NEXT]) NEQ 0 )
;  739          2460  2 DO
;  740          2461  3         IF (.MC_PTR [MC_IN_RST])
;  741          2462  3         THEN
;  742          2463  3                 IF (NOT PAT$ADD_MODULE( .MC_PTR ))              ! Try to add the module.
;  743          2464  3                 THEN
;  744          2465  3                         BEGIN
;  745          2466  3                         not_all_added = true;
;  746          2467  3                         MC_PTR [MC_IN_RST] = FALSE;
;  747          2468  3                         END;
;  748          2469  2
;  749          2470  2 !++
;  750          2471  2 ! Produce a warning message if we didn't add all those we tried to.
;  751          2472  2 !--
;  752          2473  3 IF (.NOT_ALL_ADDED)
;  753          2474  2 THEN
;  754          2475  2         SIGNAL(PAT$_NOTALLSYM);
;  755          2476  2
;  756          2477  2 RETURN(TRUE);
;  757          2478  1 END;


                                        001C 00000 BUILD_RST:
                                                           .WORD   Save R2,R3,R4                           ; 2399
                          54 00000000G   EF  9E 00002              MOVAB   PAT$GL_RST_BEGN, R4             ;
                          53             D4  00009              CLRL    NOT_ALL_ADDED                   ; 2451
                          52 00000000G   EF  D0 0000B              MOVL    PAT$GL_MC_PTR, MC_PTR           ; 2458
                50        52             64  C1 00012 1$:        ADDL3   PAT$GL_RST_BEGN, MC_PTR, R0     ; 2459
                          52             60  3C 00016              MOVZWL  (R0), MC_PTR                    ;
                                         22  13 00019              BEQL    2$                              ;
                50        52             64  C1 0001B              ADDL3   PAT$GL_RST_BEGN, MC_PTR, R0     ; 2461
                EE    03  A0             01  E1 0001F              BBC     #1, 3(R0), 1$                   ;
                                         52  DD 00024              PUSHL   MC_PTR                          ; 2463
                00000000V EF             01  FB 00026              CALLS   #1, PAT$ADD_MODULE              ;
                          E2             50  E8 0002D              BLBS    R0, 1$                          ;
                          53             01  D0 00030              MOVL    #1, NOT_ALL_ADDED               ; 2466
                50        52             64  C1 00033              ADDL3   PAT$GL_RST_BEGN, MC_PTR, R0     ; 2467
                      03  A0             02  8A 00037              BICB2   #2, 3(R0)                       ;
                                         D5  11 0003B              BRB     1$                              ; 2461
                          0D             53  E9 0003D 2$:         BLBC    NOT_ALL_ADDED, 3$               ; 2473
                             006D8013    8F  DD 00040              PUSHL   #717621T                        ; 2475
                00000000G 00             01  FB 00046              CALLS   #1, LIB$SIGNAL                  ;
                          50             01  D0 0004D 3$:         MOVL    #1, R0                          ; 2477
                                         04  00050              RET                                     ; 2478

; Routine Size:  81 bytes,    Routine Base:  _PAT$CODE + 0213
```

```
;  759          2479  1 GLOBAL ROUTINE PAT$ADD_MODULE( MODULE_TO_ADD ) =
;  760          2480  1
;  761          2481  1 !++
;  762          2482  1 ! FUNCTIONAL DESCRIPTION:
;  763          2483  1 !
;  764          2484  1 !       Add the indicated module to the RST data base.
;  765          2485  1 !
;  766          2486  1 ! FORMAL PARAMETERS:
;  767          2487  1 !
;  768          2488  1 !       MODULE_TO_ADD   -An RST-pointer to the MC record for
;  769          2489  1 !                        the module we are to initialize.
;  770          2490  1 !                        This is necessary because this module
;  771          2491  1 !                        is scope-wise 'above' all symbols
;  772          2492  1 !                        declared in the outer block of this module.
;  773          2493  1 !
;  774          2494  1 ! IMPLICIT INPUTS:
;  775          2495  1 !
;  776          2496  1 !       none.
;  777          2497  1 !
;  778          2498  1 ! IMPLICIT OUTPUTS:
;  779          2499  1 !
;  780          2500  1 !       none
;  781          2501  1 !
;  782          2502  1 ! COMPLETION CODES:
;  783          2503  1 !
;  784          2504  1 !       TRUE, if all goes OK, FALSE otherwise.
;  785          2505  1 !
;  786          2506  1 ! SIDE EFFECTS:
;  787          2507  1 !
;  788          2508  1 !       The symbols for the indicated module get added to
;  789          2509  1 !       the various RST data structures.
;  790          2510  1 !
;  791          2511  1 !--
;  792          2512  1
;  793          2513  2 BEGIN
;  794          2514  2
;  795          2515  2 LABEL
;  796          2516  2         ADD_DST_LOOP;                                   ! The overall structure of this routine
;  797          2517  2                                                         ! is one large loop in which we add
;  798          2518  2                                                         ! successive DSTs to the RST.
;  799          2519  2
;  800          2520  2 MAP
;  801          2521  2         MODULE_TO_ADD : REF MC_RECORD;
;  802          2522  2
;  803          2523  2 LOCAL
;  804          2524  2         DST_REC_ID,                                     ! ID we are passed back so that we can
;  805          2525  2                                                         !  later refer to DST records.
;  806          2526  2         DST_RECRD : REF DST_RECORD,                     ! Pointer to where a fetched DST record live
;  807          2527  2         NT_PTR : REF NT_RECORD;                         ! Pointer to NT records we build.
;  808          2528  2
;  809          2529  2 !++
;  810          2530  2 ! A new module must define a new position from which to start as far as building
;  811          2531  2 ! in scope structure is concerned.  To do this we reinitialize the scope stack,
;  812          2532  2 ! a stack of RST-pointers, the top one of which points to the NT or MC record
;  813          2533  2 ! that corresponds to the symbol (MODULE or ROUTINE) that is scope-wise above
;  814          2534  2 ! any symbol currently being added to the RST data base.  The reason why we
;  815          2535  2 ! need a stack to keep track of this is simply because BLISS allows nested
```

```
;   816          2536  2 ! ROUTINE declarations.
;   817          2537  2 !
;   818          2538  2 ! Likewise we have to stack SAT pointers to ROUTINE SAT records so that when
;   819          2539  2 ! we get the corresponding end-of-routine we can go to the SAT and fix up
;   820          2540  2 ! the UB value.
;   821          2541  2 !--
;   822          2542  2 LOCAL
;   823          2543  2         SCOPE_STACK : VECTOR [MAX_SCOPE_DEPTH +1, %SIZE(RST_POINTER)],
;   824          2544  2         SAT_RTN_STACK : VECTOR [MAX_SCOPE_DEPTH +1, %SIZE(SAT_POINTER)],
;   825          2545  2         SCOPE_STACK_PTR,                                    ! Index to current top of scope stack.
;   826          2546  2         CURENT_LANGUAGE;                                    ! Per-module source code language indicator.
;   827          2547  2
;   828          2548  2 !++
;   829          2549  2 ! Make sure that we can go ahead and add the indicated module.  Also peform
;   830          2550  2 ! any needed checks and initializations.
;   831          2551  2 !--
;   832          2552  3 IF (NOT OK_TO_ADD(.MODULE_TO_ADD, CURENT_LANGUAGE))
;   833          2553  2 THEN
;   834          2554  2         RETURN(FALSE);
;   835          2555  2
;   836          2556  2 !++
;   837          2557  2 ! Initialize the scope and SAT_RTN stacks.  (See above). Since the top-of-stack
;   838          2558  2 ! entry is MODULE, the corresponding entry in the SAT_RTN stack is not used,
;   839          2559  2 ! in BLISS, always used, in FORTRAN, and never used, in MARS.  This is because
;   840          2560  2 ! BLISS has 'real' scoping, FORTRAN's scope is only 1 level deep, and MARS
;   841          2561  2 ! doesn't give us DST entries for end-of-routines.
;   842          2562  2 !--
;   843          2563  2 SCOPE_STACK_PTR = 0;
;   844          2564  2 CH$FILL(0, %ALLOCATION(SAT_RTN_STACK), SAT_RTN_STACK );
;   845          2565  2 CH$FILL(0, %ALLOCATION(SCOPE_STACK), SCOPE_STACK );
;   846          2566  2 SCOPE_STACK[0] = .MODULE_TO_ADD;
;   847          2567  2
;   848          2568  2 !++
;   849          2569  2 ! Now process the DST record by record, ending when we encounter a module_end
;   850          2570  2 ! record, or when some error occurs.
;   851          2571  2 !--
;   852          2572  3 WHILE( (DST_RECRD = PAT$GET_NXT_DST( DST_REC_ID )) NEQ 0 )
;   853          2573  2 DO
;   854          2574  2     ADD_DST_LOOP:                                           ! Failure to add a DST => we LEAVE add block
;   855          2575  3         BEGIN
;   856          2576  3         !++
;   857          2577  3         ! We process each record depending on its DST type.
;   858          2578  3         !--
;   859          2579  3         CASE .DST_RECRD [DSTR_TYPE] FROM DST_DST_LOWEST TO DST_DST_HIGHEST OF
;   860          2580  3             SET
;   861          2581  3
;   862          2582  3             [DSC$K_DTYPE_EOM]:                              ! End module record.
;   863          2583  4                 BEGIN
;   864          2584  4                 !++
;   865          2585  4                 ! This is the ONLY successful way this routine can terminate
;   866          2586  4                 ! since we insist that a group of DST records end with an End
;   867          2587  4                 ! Module record.
;   868          2588  4                 ! Modules added OK are not about to be deleted.
;   869          2589  4                 !--
;   870          2590  4                 MODULE_TO_ADD [MC_IS_DYING] = FALSE;
;   871          2591  4
;   872          2592  4                 !++
```

```
;  873          2593  4                ! See if we can free up some of the vector storage allocated
;  874          2594  4                ! for this module.  This will be the case when we didn't actually
;  875          2595  4                ! add as many of the NTs, SATs, or LVTs as the MC statictics
;  876          2596  4                ! indicated we might. (Which is almost always the case because
;  877          2597  4                ! the gathering  is not as detailed as it might be).
;  878          2598  4                !--
;  879          2599  4                PAT$VS_SHRINK( MODULE_TO_ADD [MC_NT_STORAGE] );
;  880          2600  4                PAT$VS_SHRINK( MODULE_TO_ADD [MC_SAT_STORAGE] );
;  881          2601  4                PAT$VS_SHRINK( MODULE_TO_ADD [MC_LVT_STORAGE] );
;  882          2602  4                RETURN(TRUE);
;  883          2603  3                END;
;  884          2604  3
;  885          2605  3            [DSC$K_DTYPE_RTN]:                          ! Begin a ROUTINE record.
;  886          2606  4                BEGIN
;  887          2607  4                !++
;  888          2608  4                ! In all languages, a routine name is at least a symbol, which
;  889          2609  4                ! must therefore appear in the NT. Add the symbol to the NT.
;  890          2610  4                !--
;  891          2611  6                IF ((NT_PTR = ADD_NT(.DST_RECRD, .SCOPE_STACK[.SCOPE_STACK_PTR])
;  892          2612  5                    ) EQL 0 )
;  893          2613  4                THEN
;  894          2614  4                        LEAVE ADD_DST_LOOP;
;  895          2615  4
;  896          2616  4                !++
;  897          2617  4                ! Then, in BLISS, each routine record also  implies that we go
;  898          2618  4                ! one level down as far as scope is concerned.  This is why we
;  899          2619  4                ! maintain the so-called scope stack.
;  900          2620  4                !--
;  901          2621  5                IF (.CURENT_LANGUAGE EQL BLISS_MODULE)
;  902          2622  4                THEN
;  903          2623  5                        BEGIN
;  904          2624  5                        !++
;  905          2625  5                        ! Check for scope stack overflow.
;  906          2626  5                        !--
;  907          2627  6                        IF ((SCOPE_STACK_PTR = .SCOPE_STACK_PTR +1) GTR MAX_SCOPE_DEPTH)
;  908          2628  5                        THEN
;  909          2629  6                                BEGIN
;  910          2630  6                                $FAO_TT_OUT('!/PAT$add_module: scope depth overflow');
;  911          2631  6                                RETURN(FALSE);
;  912          2632  5                                END;
;  913          2633  5
;  914          2634  5                        !++
;  915          2635  5                        ! Push the new scope level onto the scope stack.
;  916          2636  5                        !--
;  917          2637  5                        SCOPE_STACK [.SCOPE_STACK_PTR] = .NT_PTR;
;  918          2638  4                        END;
;  919          2639  4
;  920          2640  4                !++
;  921          2641  4                ! Routine symbols also go into the SAT. The catch here is that
;  922          2642  4                ! we don't yet know the length of the routine, so we have to
;  923          2643  4                ! stack a pointer to the SAT record we build so that we can
;  924          2644  4                ! fix up this problem when we do get the info.
;  925          2645  4                !--
;  926          2646  6                IF ((SAT_RTN_STACK [.SCOPE_STACK_PTR] =
;  927          2647  6                        ADD_SAT( .NT_PTR, .DST_RECRD [DSTR_VALUE], 0))
;  928          2648  5                    EQL 0 )
;  929          2649  4                THEN
```

```
;  930     2650  5                    BEGIN
;  931     2651  5                    !++
;  932     2652  5                    ! Don't allow the next end_of_routine record
;  933     2653  5                    ! to overwrite something invalid.
;  934     2654  5                    !--
;  935     2655  5                    SAT_RTN_STACK [.SCOPE_STACK_PTR] = 0;
;  936     2656  5                    RETURN(FALSE);
;  937     2657  4                    END;
;  938     2658  3                END;
;  939     2659  3
;  940     2660  3            [DSC$K_DTYPE_EOR]:                          ! End of (BLISS and FORTRAN) routine.
;  941     2661  4                BEGIN
;  942     2662  4                !++
;  943     2663  4                ! This is where we finally get to find out the length that the
;  944     2664  4                ! current routine was.  This info allows us to complete the
;  945     2665  4                ! building of the RST record which corresponds to the routine
;  946     2666  4                ! itself.  If we do not have a valid pointer to this routine's
;  947     2667  4                ! SAT, some logic error has occurred.
;  948     2668  4                !--
;  949     2669  5                IF (.SAT_RTN_STACK [.SCOPE_STACK_PTR] NEQ 0)
;  950     2670  4                THEN
;  951     2671  5                        BEGIN
;  952     2672  5                        LOCAL
;  953     2673  5                                FIXUP_SAT_PTR : REF SAT_RECORD,
;  954     2674  5                                RTN_NT_PTR : REF NT_RECORD;
;  955     2675  5
;  956     2676  5                        !++
;  957     2677  5                        ! Pickup a pointer to the SAT record which corresponds
;  958     2678  5                        ! the associated  ROUTINE NT record.
;  959     2679  5                        !--
;  960     2680  5                        FIXUP_SAT_PTR = .SAT_RTN_STACK [.SCOPE_STACK_PTR];
;  961     2681  5
;  962     2682  5                        !++
;  963     2683  5                        ! The upper bound for this routine is the address at
;  964     2684  5                        ! which the code for it ends.  NOTE: we have now
;  965     2685  5                        ! 'bounded' the symbol.
;  966     2686  5                        !--
;  967     2687  5                        FIXUP_SAT_PTR [SAT_UB] = .FIXUP_SAT_PTR [SAT_LB]
;  968     2688  5                                + .DST_RECRD [DSTR_VALUE] -1;
;  969     2689  5                        RTN_NT_PTR = .FIXUP_SAT_PTR [SAT_NT_PTR];       ! ++37
;  970     2690  5                        RTN_NT_PTR [NT_IS_BOUNDED] = TRUE;              ! ++37
;  971     2691  5                        SAT_RTN_STACK [.SCOPE_STACK_PTR] = 0;
;  972     2692  5                        END
;  973     2693  4                ELSE
;  974     2694  5                        BEGIN
;  975     2695  5                        $FAO_TT_OUT('!/rtn end without begin');
;  976     2696  5                        RETURN(FALSE);
;  977     2697  4                        END;
;  978     2698  4
;  979     2699  4                !++
;  980     2700  4                ! Pop a scope level off of the scope stack.
;  981     2701  4                !--
;  982     2702  5                IF (.CURENT_LANGUAGE EQL BLISS_MODULE)
;  983     2703  4                THEN
;  984     2704  5                        IF ((SCOPE_STACK_PTR = .SCOPE_STACK_PTR -1) LSS 0) ! Check for scope stack underflow
;  985     2705  4                        THEN
;  986     2706  5                                BEGIN
```

```
;  987          2707  5                        $FAO_TT_OUT('!/add_module: scope depth underflow');
;  988          2708  5                        RETURN(FALSE);
;  989          2709  4                        END;
;  990          2710  3                    END;
;  991          2711  3
;  992          2712  3                [DSC$K_DTYPE_FLD]:                          ! BLISS fields are ignored.
;  993          2713  4                    BEGIN
;  994          2714  4                    TRUE;
;  995          2715  3                    END;
;  996          2716  3
;  997          2717  3                [DSC$K_DTYPE_LBL,                           ! Label or Literal DSTs. (MARS only)
;  998          2718  3                 DSC$K_DTYPE_SLB]:                          ! Labels in FORTRAN and BLISS.
;  999          2719  4                    BEGIN
; 1000          2720  4                    !++
; 1001          2721  4                    ! Add the symbol to the NT.
; 1002          2722  4                    !--
; 1003          2723  5                    IF ((NT_PTR = ADD_NT(.DST_RECRD, .SCOPE_STACK[.SCOPE_STACK_PTR])) EQL 0)
; 1004          2724  4                    THEN
; 1005          2725  4                            LEAVE ADD_DST_LOOP;
; 1006          2726  4
; 1007          2727  4                    !++
; 1008          2728  4                    ! Then, add it to the LVT or SAT, depending
; 1009          2729  4                    ! on whether the symbol is a literal or not.
; 1010          2730  4                    ! Here PATCH ignores registers while DEBUG does not.
; 1011          2731  4                    ! Hence, the more specific check of field DSTR_ACC01.
; 1012          2732  4                    !--
; 1013          2733  5                    IF ((.DST_RECRD [DSTR_TYPE] EQL DSC$K_DTYPE_SLB) OR
; 1014          2734  5                       (.DST_RECRD [DSTR_ACC01] EQL ACCS_VALUEADR))
; 1015          2735  4                    THEN
; 1016          2736  5                            BEGIN
; 1017          2737  6                            IF (ADD_SAT( .NT_PTR, .DST_RECRD [DSTR_VALUE], 0) EQL 0)
; 1018          2738  5                            THEN
; 1019          2739  5                                    RETURN(FALSE);
; 1020          2740  5                            END
; 1021          2741  4                    ELSE
; 1022          2742  5                            BEGIN
; 1023          2743  6                            IF (.DST_RECRD [DSTR_ACC01] EQL ACCS_LITERAL)
; 1024          2744  5                            THEN
; 1025          2745  6                                    IF (NOT ADD_LVT( .NT_PTR, .DST_RECRD [DSTR_VALUE]))
; 1026          2746  5                                    THEN
; 1027          2747  5                                            RETURN(FALSE);
; 1028          2748  4                            END;
; 1029          2749  3                    END;
; 1030          2750  3                [DSC$K_DTYPE_PCT]:                          ! P-SECT record.
; 1031          2751  4                    BEGIN
; 1032          2752  4                    LOCAL
; 1033          2753  4                            PSECT_LENGTH : REF VECTOR[,LONG];
; 1034          2754  4                    !++
; 1035          2755  4                    ! Add the symbol to the NT.
; 1036          2756  4                    !--
; 1037          2757  5                    IF ((NT_PTR = ADD_NT(.DST_RECRD, .SCOPE_STACK[.SCOPE_STACK_PTR])) EQL 0)
; 1038          2758  4                    THEN
; 1039          2759  4                            LEAVE ADD_DST_LOOP;
; 1040          2760  4
; 1041          2761  4                    !++
; 1042          2762  4                    ! The symbols also goes into the SAT. Since we have to pass on
; 1043          2763  4                    ! the length of the PSECT, we must extract a field which is
```

```
;1044     2764  4                 ! in a variable position within the DST. This code must be
;1045     2765  4                 ! changed if the psect length is not directly after the counted
;1046     2766  4                 ! string name.
;1047     2767  4                 !--
;1048     2768  4                 PSECT_LENGTH = DST_RECRD[DSTR_NAME] + .DST_RECRD[DSTR_NAME] +1;
;1049     2769  5                 IF (ADD_SAT( .NT_PTR, .DST_RECRD [DSTR_VALUE],
;1050     2770  5                         .DST_RECRD [DSTR_VALUE] + .PSECT_LENGTH[0] ) EQL 0)
;1051     2771  4                 THEN
;1052     2772  4                         RETURN(FALSE);
;1053     2773  3                 END;
;1054     2774  3
;1055     2775  3             [INRANGE, OUTRANGE]:
;1056     2776  4                 BEGIN
;1057     2777  4                 !++
;1058     2778  4                 ! This should be a separate CASE, but isn't due simply to the
;1059     2779  4                 ! size of the case table.  PATCH handles the following types
;1060     2780  4                 ! of symbols:
;1061     2781  4                 !       dsc$k_dtype_v           bit
;1062     2782  4                 !       dsc$k_dtype_bu          byte logical
;1063     2783  4                 !       dsc$k_dtype_wu          word logical
;1064     2784  4                 !       dsc$k_dtype_lu          longword logical
;1065     2785  4                 !       dsc$k_dtype_qu          quadword logical
;1066     2786  4                 !       dsc$k_dtype_b           byte integer
;1067     2787  4                 !       dsc$k_dtype_w           word integer
;1068     2788  4                 !       dsc$k_dtype_l           longword integer
;1069     2789  4                 !       dsc$k_dtype_q           quadword integer
;1070     2790  4                 !       dsc$k_dtype_f           single-precision floating
;1071     2791  4                 !       dsc$k_dtype_d           double-precision floating
;1072     2792  4                 !       dsc$k_dtype_fc          complex
;1073     2793  4                 !       dsc$k_dtype_dc          double-precision complex
;1074     2794  4                 !       dsc$k_dtype_t           ascii text
;1075     2795  4                 !       dsc$k_dtype_nu          numeric string, unsigned
;1076     2796  4                 !       dsc$k_dtype_nl          numeric string, left separate sign
;1077     2797  4                 !       dsc$k_dtype_nlo         numeric string, left overpunched sign
;1078     2798  4                 !       dsc$k_dtype_nr          numeric string, right separate sign
;1079     2799  4                 !       dsc$k_dtype_nro         numeric string, right overpunched sign
;1080     2800  4                 !       dsc$k_dtype_nz          numeric string, zoned sign
;1081     2801  4                 !       dsc$k_dtype_p           packed decimal string
;1082     2802  4                 !       dsc$k_dtype_zi          sequence of instructions
;1083     2803  4                 !       dsc$k_dtype_zem         procedure entry mask
;1084     2804  4                 ! PATCH ignores the following types of symbols:
;1085     2805  4                 !       dsc$k_dtype_z           unspecified
;1086     2806  4                 !--
;1087     2807  4                 IF (.DST_RECRD [DSTR_TYPE] GEQ DSC$K_DTYPE_V) AND
;1088     2808  5                    (.DST_RECRD [DSTR_TYPE] LEQ DST_TYP_HIGHEST)
;1089     2809  4                 THEN
;1090     2810  4                         !++
;1091     2811  4                         ! Check that PATCH can handle the symbol.  Discard
;1092     2812  4                         ! symbols defined as registers or by descriptors,
;1093     2813  4                         ! (bits 2+3 equal to 2 or 3) and symbols whose values
;1094     2814  4                         ! are based off registers (bits 0+1 equal to 2 or 3).
;1095     2815  4                         ! The remaining symbols (literals or value addresses not
;1096     2816  4                         ! based off registers) are now added to the NT and SAT.
;1097     2817  4                         !--
;1098     2818  5                         IF ((.DST_RECRD[DSTR_ACC01] EQL ACCS_LITERAL) OR
;1099     2819  5                             (.DST_RECRD[DSTR_ACC01] EQL ACCS_VALUEADR))
;1100     2820  4                         AND
```

```
;  1101          2821  5                ((.DST_RECRD[DSTR_ACC23] EQL ACCS_NOTBASDIR) OR
;  1102          2822  5                 (.DST_RECRD[DSTR_ACC23] EQL ACCS_NOTBASIND))
;  1103          2823  4            THEN
;  1104          2824  5                    BEGIN
;  1105          2825  5                    !++
;  1106          2826  5                    ! Most of these symbols should also go into the SAT.
;  1107          2827  5                    ! The ones that should not are the BLISS type 0
;  1108          2828  5                    ! symbols which are not static.  PATCH
;  1109          2829  5                    ! completely ignores all records which are type
;  1110          2830  5                    ! 0 (unspecified) so there is no test to weed
;  1111          2831  5                    ! them out here.
;  1112          2832  5                    ! Processing of SRM data types depends
;  1113          2833  5                    ! on whether the datum is scalar or not.
;  1114          2834  5                    ! Only scalars are handled here.
;  1115          2835  5                    !--
;  1116          2836  5                    LOCAL
;  1117          2837  5                        SIZE;
;  1118          2838  5                    OWN
;  1119          2839  5                        SRM_LENGTHS : VECTOR[12,BYTE]       ! Sizes of SRM type data.
;  1120          2840  5                        INITIAL( BYTE (
;  1121          2841  5                            1, !    DSC$K_DTYPE_BU, Byte logical.
;  1122          2842  5                            2, !    DSC$K_DTYPE_WU, Word logical.
;  1123          2843  5                            4, !    DSC$K_DTYPE_LU, Longword logical.
;  1124          2844  5                            8, !    DSC$K_DTYPE_QU, Quadword logical.
;  1125          2845  5                            1, !    DSC$K_DTYPE_B,  Byte integer.
;  1126          2846  5                            2, !    DSC$K_DTYPE_W,  Word integer.
;  1127          2847  5                            4, !    DSC$K_DTYPE_L,  Longword integer.
;  1128          2848  5                            8, !    DSC$K_DTYPE_Q,  Quadword integer.
;  1129          2849  5                            4, !    DSC$K_DTYPE_F,  single-precision floating
;  1130          2850  5                            8, !    DSC$K_DTYPE_D,  double-precision floating
;  1131          2851  5                            8, !    DSC$K_DTYPE_FC, complex
;  1132          2852  5                            16 !    DSC$K_DTYPE_DC, double-precision complex
;  1133          2853  5                        ));
;  1134          2854  5
;  1135          2855  5                    !++
;  1136          2856  5                    ! The only so-called SRM types which we
;  1137          2857  5                    ! know the size of are those from
;  1138          2858  5                    ! DSC$k_DTYPE_BU to _DC, inclusive.
;  1139          2859  5                    ! We simply pick this size value out
;  1140          2860  5                    ! of the above vector.
;  1141          2861  5                    !--
;  1142          2862  6                    IF ((NT_PTR = ADD_NT(.DST_RECRD, .SCOPE_STACK[.SCOPE_STACK_PTR])) EQL 0)
;  1143          2863  5                    THEN
;  1144          2864  5                        LEAVE ADD_DST_LOOP;
;  1145          2865  5                    SIZE = .DST_RECRD[DSTR_TYPE];
;  1146          2866  5                    IF (.SIZE GEQ DSC$K_DTYPE_BU) AND
;  1147          2867  6                        (.SIZE LEQ DSC$K_DTYPE_DC)
;  1148          2868  5                    THEN
;  1149          2869  5                        SIZE = .DST_RECRD[DSTR_VALUE] - 1 +
;  1150          2870  5                               .SRM_LENGTHS[.SIZE - DSC$K_DTYPE_BU]
;  1151          2871  5                    ELSE
;  1152          2872  5                        !++
;  1153          2873  5                        ! Unknown upper-bound addresses
;  1154          2874  5                        ! must be 0.
;  1155          2875  5                        !--
;  1156          2876  5                        SIZE = 0;
;  1157          2877  5
```

```
;  1158     2878  5                    !++
;  1159     2879  5                    ! Add the symbols known to be static to the SAT.
;  1160     2880  5                    !--
;  1161     2881  6                    IF (ADD_SAT(.NT_PTR, .DST_RECRD [DSTR_VALUE], .SIZE) EQL 0)
;  1162     2882  5                    THEN
;  1163     2883  5                        RETURN(FALSE);
;  1164     2884  5                    END                                 ! end of TYPES _V THRU _Q
;  1165     2885  4                ELSE
;  1166     2886  5                    BEGIN
;  1167     2887  5                    !++
;  1168     2888  5                    ! Array types are more complicated than scalars
;  1169     2889  5                    ! as there is more involved in calculating the
;  1170     2890  5                    ! lower and upper address bounds.
;  1171     2891  5                    !--
;  1172     2892  5                    LOCAL
;  1173     2893  5                        DESC_ADDRESS,
;  1174     2894  5                        BOUNDS : ARRAY_BNDS_DESC;
;  1175     2895  5                    !++
;  1176     2896  5                    ! Eliminate all dynamic arrays.  Those kept are
;  1177     2897  5                    ! are described within the DST -- so called
;  1178     2898  5                    ! "PC" based descriptors which have _BASD = 2
;  1179     2899  5                    ! and _BREG = 15.
;  1180     2900  5                    !--
;  1181     2901  5                    IF (.DST_RECRD[DSTR_ACCES_TYPE] EQL ACCS_DESCRIPTOR) AND
;  1182     2902  5                       (.DST_RECRD[DSTR_ACCES_BASD] EQL ACCS_BASDIR) AND
;  1183     2903  6                          (.DST_RECRD[DSTR_ACCES_BREG] EQL T5)
;  1184     2904  5                    THEN
;  1185     2905  6                        BEGIN
;  1186     2906  6                        !++
;  1187     2907  6                        ! If _BASD and _BREG are both zero, then
;  1188     2908  6                        ! the descriptor may not be complete.
;  1189     2909  6                        ! In this case the ADD_NT call fails.
;  1190     2910  6                        !--
;  1191     2911  7                        IF ((NT_PTR = ADD_NT(.DST_RECRD, .SCOPE_STACK[.SCOPE_STACK_PTR])) EQ
;  1192     2912  6                        THEN
;  1193     2913  6                            LEAVE ADD_DST_LOOP;
;  1194     2914  6                        !++
;  1195     2915  6                        ! Get the address of the array descriptor and
;  1196     2916  6                        ! use it to build a 'bounds descriptor'.
;  1197     2917  6                        !--
;  1198     2918  7                        IF (PAT$DST_VALUE(.DST_RECRD, DESC_ADDRESS) EQL 0)
;  1199     2919  6                        THEN
;  1200     2920  6                            RETURN(FALSE);          ! THIS TYPE NOT HANDLED
;  1201     2921  6                        PAT$GET_BOUNDS(.DESC_ADDRESS, BOUNDS);
;  1202     2922  6
;  1203     2923  6                        !++
;  1204     2924  6                        ! The beginning of the array plus the number of
;  1205     2925  6                        ! bytes of elements minus one is the address of
;  1206     2926  6                        ! the end of the array.
;  1207     2927  6                        ! Add this symbol to the SAT.
;  1208     2928  6                        !--
;  1209     2929  7                        IF (ADD_SAT(.NT_PTR, .BOUNDS[ARRAY_ADDRESS],
;  1210     2930  7                                    .BOUNDS[ARRAY_ADDRESS] +
;  1211     2931  7                                    .BOUNDS[ARRAY_LENGTH] -1)
;  1212     2932  7                           EQL 0)
;  1213     2933  6                        THEN
;  1214     2934  6                            RETURN(FALSE);
```

```
;  1215          2935  5                                   END;                    ! END OF ARRAYS HANDLED CURRENTLY
;  1216          2936  4                               END;                        ! END OF ARRAYS AND DESCRIPTORS
;  1217          2937  3                           END;                            ! END OF INRANGE,OUTRANGE
;  1218          2938  3                       TES;                                ! END OF CASE LOOP
;  1219          2939  2                   END;                                    ! Go back and process the next record. (ADD_
;  1220          2940  2
;  1221          2941  2 !++
;  1222          2942  2 ! If the above WHILE exits, then we encountered the end of the DST records
;  1223          2943  2 ! before we got an END_MODULE record to end the beginning MODULE one.  This is
;  1224          2944  2 ! considered a fatal error.
;  1225          2945  2 !--
;  1226          2946  2 $FAO_TT_OUT('!/Premature end of DST in routine ADD_MODULE.');
;  1227          2947  2
;  1228          2948  2 RETURN(FALSE);
;  1229          2949  1 END;
;  INFO#212                L1:2630
;  Null expression appears in value-required context
;  INFO#212                L1:2695
;  Null expression appears in value-required context
;  INFO#212                L1:2707
;  Null expression appears in value-required context
;  INFO#212                L1:2946
;  Null expression appears in value-required context


                                                                .PSECT  _PAT$PLIT,NOWRT,NOEXE,0

                                                        26  00087 P.AAF:   .BYTE   38
6C 75 64 6F 6D 5F 64 64 61 24 54 41 50 2F               21  00088          .ASCII  \!/PAT$add_module: scope depth overflow\
20 68 74 70 65 64 20 65 70 6F 63 73 20 3A               65  00097
                    77 6F 6C 66 72 65 76                6F  000A6
                                                        17  000AE P.AAG:   .BYTE   23
6F 68 74 69 77 20 64 6E 65 20 6E 74 72 2F               21  000AF          .ASCII  \!/rtn end without begin\
                    6E 69 67 65 62 20 74                75  000BE
                                                        23  000C6 P.AAH:   .BYTE   35
73 20 3A 65 6C 75 64 6F 6D 5F 64 64 61 2F               21  000C7          .ASCII  \!/add_module: scope depth underflow\
65 64 6E 75 20 68 74 70 65 64 20 65 70 6F               63  000D6
                                    77 6F 6C 66         72  000E5
                                                        2D  000EA P.AAI:   .BYTE   45
64 6E 65 20 65 72 75 74 61 6D 65 72 50 2F               21  000EB          .ASCII  \!/Premature end of DST in routine ADD_MO\
74 75 6F 72 20 6E 69 20 54 53 44 20 66 6F               20  000FA
                    4F 4D 5F 44 44 41 20 65 6E          69  00109
                                    2E 45 4C 55 44      00113          .ASCII  \DULE.\

                                                                .PSECT  _PAT$OWN,NOEXE,2

                10 08 08 04 08 04 02 01 08 04 02 01     00004 SRM_LENGTHS:
                                                                .BYTE   1, 2, 4, 8, 1, 2, 4, 8, 4, 8, 8, 16             ;


                                                                .PSECT  _PAT$CODE,NOWRT,2

                                                        0FFC 00000      .ENTRY  PAT$ADD_MODULE, Save R2,R3,R4,R5,R6,R7,R8,- ; 2479
                                                                        R9,R10,R11
                                        5B 00000000G    EF   9E 00002   MOVAB   PAT$VS_SHRINK, R11                       ;
```

```
                                5A 00000000'   EF   9E 00009          MOVAB   P.AAF, R10                                                    ;
                                59 00000000G   EF   9E 00010          MOVAB   PAT$GL_RST_BEGN, R9                                          ;
                                58 00000000V   EF   9E 00017          MOVAB   ADD_NT, R8                                                   ;
                                5E        A8   AE   9E 0001E          MOVAB   -88(SP), SP                                                  ;
                                               5E   DD 00022          PUSHL   SP                                                           ; 2552
                                57        04   AC   D0 00024          MOVL    MODULE_TO_ADD, R7                                            ;
                                               57   DD 00028          PUSHL   R7                                                           ;
                          00000000V   EF        02   FB 0002A          CALLS   #2, OK_TO_ADD                                                ;
                                03              50   E8 00031          BLBS    R0, 1$                                                       ;
                                            020A   31 00034          BRW     34$                                                          ;
                                               56   D4 00037 1$:      CLRL    SCOPE_STACK_PTR                                              ; 2563
                    2C          00        6E        00   2C 00039          MOVC5   #0, (SP), #0, #44, SAT_RTN_STACK                     ; 2564
                                        14   AE             0003E
                    16          00        6E        00   2C 00040          MOVC5   #0, (SP), #0, #22, SCOPE_STACK                       ; 2565
                                        40   AE             00045
                                   40   AE             57   B0 00047          MOVW    R7, SCOPE_STACK                                      ; 2566
                                        04   AE             9F 0004B 2$:      PUSHAB  DST_REC_ID                                           ; 2572
                          00000000G   EF             01   FB 0004E          CALLS   #1, PAT$GET_NXT_DST                                  ;
                                52              50   D0 00055          MOVL    R0, DST_RECRD                                                ;
                                               03   12 00058          BNEQ    3$                                                           ;
                                            01D8   31 0005A          BRW     32$                                                          ;
                          08        B7   8F        01   A2 8F 0005D 3$:      CASEB   1(DST_RECRD), #183, #8                               ; 2579
                    016A          0012        01A3        FFE8 00063 4$:      .WORD   2$-4$,-                                              ;
                    00EB          00C4        0012        016A 0006B          29$-4$,-                                                     ;
                                              0012        012C 00073          5$-4$,-                                                      ;
                                                                              25$-4$,-                                                     ;
                                                                              25$-4$,-                                                     ;
                                                                              5$-4$,-                                                      ;
                                                                              15$-4$,-                                                     ;
                                                                              16$-4$,-                                                     ;
                                                                              19$-4$                                                       ;
                                         01    A2   95 00075 5$:      TSTB    1(DST_RECRD)                                                 ; 2807
                                               D1   13 00078          BEQL    2$                                                           ;
                                      17       01    A2   91 0007A          CMPB    1(DST_RECRD), #23                                    ; 2808
                                               CB   1A 0007E          BGTRU   2$                                                           ;
                    50        02    A2        02        00   EF 00080          EXTZV   #0, #2, 2(DST_RECRD), R0                             ; 2818
                                               05   13 00086          BEQL    6$                                                           ;
                                      01       50   D1 00088          CMPL    R0, #1                                                       ; 2819
                                               44   12 000BA          BNEQ    10$                                                          ;
                                      0C       02    A2   93 0008  6$:      BITB    2(DST_RECRD), #12                                    ; 2821
                                               08   13 00091          BEQL    7$                                                           ;
                    01        02    A2        02        02   ED 00093          CMPZV   #2, #2, 2(DST_RECRD), #1                             ; 2822
                                               36   12 00099          BNEQ    10$                                                          ;
                                      7E       40 AE46   3C 0009B 7$:      MOVZWL  SCOPE_STACK[SCOPE_STACK_PTR], -(SP)                  ; 2862
                                               52   DD 000A0          PUSHL   DST_RECRD                                                    ;
                                      68              02   FB 000A2          CALLS   #2, ADD_NT                                           ;
                                      53              50   D0 000A5          MOVL    R0, NT_PTR                                           ;
                                               A1   13 000A8          BEQL    2$                                                           ;
                                      50       01    A2   9A 000AA          MOVZBL  1(DST_RECRD), SIZE                                   ; 2865
                                      02              50   D1 000AE          CMPL    SIZE, #2                                             ; 2866
                                               17   19 000B1          BLSS    8$                                                           ;
                                      0D              50   D1 000B3          CMPL    SIZE, #13                                            ; 2867
                                               12   14 000B6          BGTR    8$                                                           ;
                                      51 00000000'EF40   9A 000B8          MOVZBL  SRM_LENGTHS-2[SIZE], R1                              ; 2870
                                      51        03    A2   C0 000C0          ADDL2   3(DST_RECRD), R1                                     ;
                                      50        FF    A1   9E 000C4          MOVAB   -1(R1), SIZE                                         ; 2869
                                               02   11 000C8          BRB     9$                                                           ;
```

```
                                                50   D4 000CA 8$:     CLRL    SIZE                                            ; 2876
                                                50   DD 000CC 9$:     PUSHL   SIZE                                            ; 2881
                                             0152   31 000CE         BRW     30$
                                          02    50   D1 000D1 10$:    CMPL    R0, #2                                          ; 2901
                                                0E   12 000D4         BNEQ    11$
                02        02    A2            02   ED 000D6         CMPZV   #2, #2, 2(DST_RECRD), #2                        ; 2902
                                                06   12 000DC         BNEQ    11$
                0F        02    A2            04   ED 000DE         CMPZV   #4, #4, 2(DST_RECRD), #15                       ; 2903
                                                03   13 000E4 11$:    BEQL    13$
                                             FF62   31 000E6 12$:    BRW     2$
                                          7E    40 AE46   3C 000E9 13$:    MOVZWL  SCOPE_STACK[SCOPE_STACK_PTR], -(SP)             ; 2911
                                                52   DD 000EE         PUSHL   DST_RECRD
                                          68    02   FB 000F0         CALLS   #2, ADD_NT
                                          53    50   D0 000F3         MOVL    R0, NT_PTR
                                                EE   13 000F6         BEQL    12$
                                             08    AE   9F 000F8         PUSHAB  DESC_ADDRESS                                    ; 2918
                                                52   DD 000FB         PUSHL   DST_RECRD
                   00000000G    EF               02   FB 000FD         CALLS   #2, PAT$DST_VALUE
                                                50   D5 00104         TSTL    R0
                                                03   12 00106         BNEQ    14$
                                             0136   31 00108         BRW     34$
                                             0C    AE   9F 0010B 14$:    PUSHAB  BOUNDS                                          ; 2921
                                             0C    AE   DD 0010E         PUSHL   DESC_ADDRESS
                   00000000V    EF               02   FB 00111         CALLS   #2, PAT$GET_BOUNDS
                50        0C    AE            10   AE   C1 00118         ADDL3   BOUNDS+4, BOUNDS, R0                            ; 2931
                                             FF    A0   9F 0011E         PUSHAB  -1(R0)
                                             10    AE   DD 00121         PUSHL   BOUNDS                                          ; 2929
                                                00FF   31 00124         BRW     1$
                50                 57               69   C1 00127 15$:    ADDL3   PAT$GL_RST_BEGN, R7, R0                         ; 2590
                              03   A0            40   8F   8A 0012B         BICB2   #64, 3(R0)
                                             1C    A0   9F 00130         PUSHAB  28(R0)                                          ; 2599
                                          6B               01   FB 00133         CALLS   #1, PAT$VS_SHRINK
                50                 57               69   C1 00136         ADDL3   PAT$GL_RST_BEGN, R7, R0                         ; 2600
                                             23    A0   9F 0013A         PUSHAB  35(R0)
                                          6B               01   FB 0013D         CALLS   #1, PAT$VS_SHRINK
                50                 57               69   C1 00140         ADDL3   PAT$GL_RST_BEGN, R7, R0                         ; 2601
                                             2A    A0   9F 00144         PUSHAB  42(R0)
                                          6B               01   FB 00147         CALLS   #1, PAT$VS_SHRINK
                                          50               01   D0 0014A         MOVL    #1, R0                                  ; 2602
                                                           04   00 0014D         RET
                                          7E    40 AE46   3C 0014E 16$:    MOVZWL  SCOPE_STACK[SCOPE_STACK_PTR], -(SP)             ; 2611
                                                52   DD 00153         PUSHL   DST_RECRD
                                          68    02   FB 00155         CALLS   #2, ADD_NT
                                          53    50   D0 00158         MOVL    R0, NT_PTR
                                                66   13 0015B         BEQL    22$                                             ; 2612
                                          02    6E   D1 0015D         CMPL    CURENT_LANGUAGE, #2                             ; 2621
                                                12   12 00160         BNEQ    18$
                                                56   D6 00162         INCL    SCOPE_STACK_PTR                                 ; 2627
                                          0A    56   D1 00164         CMPL    SCOPE_STACK_PTR, #10
                                                06   15 00167         BLEQ    17$
                                                7E   D4 00169         CLRL    -(SP)                                           ; 2630
                                                5A   DD 0016B         PUSHL   R10
                                                5C   11 0016D         BRB     24$
                40 AE46                         53   B0 0016F 17$:    MOVW    NT_PTR, SCOPE_STACK[SCOPE_STACK_PTR]            ; 2637
                                                7E   D4 00174 18$:    CLRL    -(SP)                                           ; 2647
                                             03    A2   DD 00176         PUSHL   3(DST_RECRD)
                                                53   DD 00179         PUSHL   NT_PTR
```

```
                00000000V  EF              03  FB 0017B          CALLS   #3, ADD_SAT                                      ;
                      14 AE46              50  D0 00182          MOVL    R0, SAT_RTN_STACK[SCOPE_STACK_PTR]               ;
                                           3A  12 00187          BNEQ    22$                                              ; 2648
                                     14 AE46  D4 00189          CLRL    SAT_RTN_STACK[SCOPE_STACK_PTR]                   ; 2655
                                           75  11 0018D          BRB     28$                                              ; 2656
                                  50       14 AE46  D0 0018F 19$: MOVL    SAT_RTN_STACK[SCOPE_STACK_PTR], R0               ; 2669
                                           1C  13 00194          BEQL    20$                                              ;
                51        02     A0        03    A2  C1 00196          ADDL3   3(DST_RECRD), 2(FIXUP_SAT_PTR), R1       ; 2688
                          06     A0        FF    A1  9E C0019C         MOVAB   -1(R1), 6(FIXUP_SAT_PTR)                 ;
                                  50              60  3C 001A1          MOVZWL  (FIXUP_SAT_PTR), RTN_NT_PTR              ; 2689
                                  50              69  C0 001A4          ADDL2   PAT$GL_RST_BEGN, R0                      ; 2690
                          03     A0              80  8F 88 001A7       BISB2   #128, 3(R0)                              ;
                                                 14 AE46  D4 001AC      CLRL    SAT_RTN_STACK[SCOPE_STACK_PTR]           ; 2691
                                                    07  11 001B0       BRB     21$                                      ; 2669
                                                    7E  D4 001B2 20$:  CLRL    -(SP)                                    ; 2695
                                                 27    AA  9F 001B4    PUSHAB  P.AAG                                    ;
                                                    12  11 001B7       BRB     24$                                      ;
                                  02                6E  D1 001B9 21$:  CMPL    CURENT_LANGUAGE, #2                      ; 2702
                                                    05  12 001BC       BNEQ    22$                                      ;
                                  02                56  F4 001BE       SOBGEQ  SCOPE_STACK_PTR, 22$                     ; 2704
                                                    03  11 001C1       BRB     23$                                      ;
                                                FE85    31 001C3 22$:  BRW     2$                                       ;
                                                    7E  D4 001C6 23$:  CLRL    -(SP)                                    ; 2707
                                                 3F    AA  9F 001C8    PUSHAB  P.AAH                                    ;
                                                    6D  11 001CB 24$:  BRB     33$                                      ;
                                  7E             40 AE46  3C 001CD 25$: MOVZWL  SCOPE_STACK[SCOPE_STACK_PTR], -(SP)      ; 2723
                                                    52  DD 001D2       PUSHL   DST_RECRD                                ;
                                  68                02  FB 001D4       CALLS   #2, ADD_NT                               ;
                                  53                50  D0 001D7       MOVL    R0, NT_PTR                               ;
                                                    E7  13 001DA       BEQL    22$                                      ;
                             BB   8F              01    A2  91 001DC    CMPB    1(DST_RECRD), #187                       ; 2733
                                                    08  13 001E1       BEQL    26$                                      ;
                01        02     A2              02    00  ED 001E3    CMPZV   #0, #2, 2(DST_RECRD), #1                 ; 2734
                                                    04  12 001E9       BNEQ    27$                                      ;
                                                    7E  D4 001EB 26$:  CLRL    -(SP)                                    ; 2737
                                                    34  11 001ED       BRB     30$                                      ;
                                  03              02    A2  93 001EF 27$: BITB    2(DST_RECRD), #3                         ; 2743
                                                    CE  12 001F3       BNEQ    22$                                      ;
                                                 03    A2  DD 001F5    PUSHL   3(DST_RECRD)                             ; 2745
                                                    53  DD 001F8       PUSHL   NT_PTR                                   ;
                          00000000V  EF              02  FB 001FA       CALLS   #2, ADD_LVT                              ;
                                     BF              50  E8 00201       BLBS    R0, 22$                                  ;
                                                    3B  11 00204 28$:  BRB     34$                                      ; 2747
                                  7E             40 AE46  3C 00206 29$: MOVZWL  SCOPE_STACK[SCOPE_STACK_PTR], -(SP)      ; 2757
                                                    52  DD 0020B       PUSHL   DST_RECRD                                ;
                                  68                02  FB 0020D       CALLS   #2, ADD_NT                               ;
                                  53                50  D0 00210       MOVL    R0, NT_PTR                               ;
                                                    AE  13 00213       BEQL    22$                                      ;
                                  50             07    A2  9A 00215    MOVZBL  7(DST_RECRD), R0                         ; 2768
                                  50             08 A042  9E 00219    MOVAB   8(R0)[DST_RECRD], PSECT_LENGTH           ;
                          7E      03              60  C1 0021E       ADDL3   (PSECT_LENGTH), 3(DST_RECRD), -(SP)      ; 2770
                                                 03    A2  DD 00223 30$: PUSHL   3(DST_RECRD)                             ; 2769
                                                    53  DD 00226 31$: PUSHL   NT_PTR                                   ;
                          00000000V  EF              03  FB 00228       CALLS   #3, ADD_SAT                              ;
                                                    50  D5 0022F       TSTL    R0                                       ; 2770
                                                    90  12 00231       BNEQ    22$                                      ;
                                                    0C  11 00233       BRB     34$                                      ; 2772
```

```
                                7E  D4 00235 32$:     CLRL    -(SP)                                   ; 2946
                             63 AA  9F 00237         PUSHAB  P.AAI                                   ;
                                                                                                     ;
                 00000000G   EF 02  FB 0023A 33$:    CALLS   #2, PAT$FAO_OUT                         ;
                                50  D4 00241 34$:     CLRL    R0                                      ; 2949
                                    04 00243         RET                                             ;

; Routine Size:  580 bytes,    Routine Base:  _PAT$CODE + 0264
```

```
;  1231     2950  1 ROUTINE OK_TO_ADD( MODULE_TO_ADD, LANGUAGE_ADDR ) =
;  1232     2951  1
;  1233     2952  1 !++
;  1234     2953  1 ! FUNCTIONAL DESCRIPTION:
;  1235     2954  1 !
;  1236     2955  1 !       See if it is OK to add the indicated module to the RST data base.
;  1237     2956  1 !       Also perform all necessary initializations so that ADD_MODULE
;  1238     2957  1 !       can go ahead an do so.
;  1239     2958  1 !
;  1240     2959  1 ! FORMAL PARAMETERS:
;  1241     2960  1 !
;  1242     2961  1 !       MODULE_TO_ADD   -An RST-pointer to the MC record for
;  1243     2962  1 !                        the module we are to initialize.
;  1244     2963  1 !       LANGUAGE_ADDR   -The address of where we should pass
;  1245     2964  1 !                        back the module's language.  If this
;  1246     2965  1 !                        value = 0, we don't position the DST
;  1247     2966  1 !                        or check or pass back the language indicator.
;  1248     2967  1 !
;  1249     2968  1 ! IMPLICIT INPUTS:
;  1250     2969  1 !
;  1251     2970  1 !       We believe the given MC pointer.  In particular,
;  1252     2971  1 !       we will not allocate storage for a module
;  1253     2972  1 !       if the corresponding storage descriptor
;  1254     2973  1 !       already contains a pointer.  (i.e. is not 0).
;  1255     2974  1 !
;  1256     2975  1 ! IMPLICIT OUTPUTS:
;  1257     2976  1 !
;  1258     2977  1 !       ADD_MODULE will not fail if it is called after this
;  1259     2978  1 !       routine successfully completes.
;  1260     2979  1 !
;  1261     2980  1 !       The language code may be passed back to the caller.
;  1262     2981  1 !
;  1263     2982  1 ! ROUTINE VALUE:
;  1264     2983  1 ! COMPLETION CODES:
;  1265     2984  1 !
;  1266     2985  1 !       TRUE, if all goes OK, FALSE otherwise.
;  1267     2986  1 !
;  1268     2987  1 ! SIDE EFFECTS:
;  1269     2988  1 !
;  1270     2989  1 !       Storage for all symbols for this module
;  1271     2990  1 !       is allocated so that PAT$VS_GET can then
;  1272     2991  1 !       simply dole it out.
;  1273     2992  1 !
;  1274     2993  1 !       The DST is 'positioned' so that PAT$GET_NXT_DST
;  1275     2994  1 !       will be able to sequentially read the DST starting
;  1276     2995  1 !       at the first record in the DST after the DST record
;  1277     2996  1 !       for the indicated module.
;  1278     2997  1 !
;  1279     2998  1 !--
;  1280     2999  1
;  1281     3000  2 BEGIN
;  1282     3001  2
;  1283     3002  2 MAP
;  1284     3003  2         LANGUAGE_ADDR : REF VECTOR,
;  1285     3004  2         MODULE_TO_ADD : REF MC_RECORD;
;  1286     3005  2
;  1287     3006  2 LOCAL
```

```
;  1288          3007  2          DST_RECRD : REF DST_RECORD,                          ! Pointer to where a fetched DST record live
;  1289          3008  2          BEGIN_DST_AT,                                        ! ID of DST record where the DST for
;  1290          3009  2                                                               !  the module we are to add begins.
;  1291          3010  2          DST_REC_ID,                                          ! ID we are passed back so that we can
;  1292          3011  2                                                               !  later refer to DST records.
;  1293          3012  2          CURENT_LANGUAGE,                                     ! Per-module source code language indicator.
;  1294          3013  2          NT_PTR : REF NT_RECORD,                              ! Pointer to NT records we build.
;  1295          3014  2          STORAGE_DESC : REF VECT_STORE_DESC;                  ! We use the MC-contained descriptor of
;  1296          3015  2                                                               ! vector storage for all indicated symbols.
;  1297          3016  2
;  1298          3017  2      !++
;  1299          3018  2      ! Make sure that we have enough space to add this module before we begin.  This
;  1300          3019  2      ! is necessary because it would be very difficult to 'back out' once we began,
;  1301          3020  2      ! and we wouldn't like to have modules half initialized.
;  1302          3021  2      !--
;  1303          3022  3      IF (NOT (PAT$REPORT_FREE() - PAT$MODULE_SIZE(.MODULE_TO_ADD)) GTR RST_AVAIL_SIZE)
;  1304          3023  2      THEN
;  1305          3024  2              !++
;  1306          3025  2              ! We don't try to handle this one here because what is appropriate
;  1307          3026  2              ! depends on when (why) the module was being initialized.
;  1308          3027  2              !--
;  1309          3028  2              RETURN(FALSE);
;  1310          3029  2
;  1311          3030  2      !++
;  1312          3031  2      ! Before allocating space for NT symbols, make sure that there will be some.
;  1313          3032  2      !  It makes no sense for a module to be added which has no name symbols.
;  1314          3033  2      !--
;  1315          3034  3      IF (.MODULE_TO_ADD [MC_NAMES] EQL 0)
;  1316          3035  2      THEN
;  1317          3036  2              RETURN(FALSE);
;  1318          3037  2      !++
;  1319          3038  2      ! See if NT storage has already been allocated.  To do this, pick up the vector
;  1320          3039  2      ! storage descriptor for the  indicated NT records and see that it points to 0.
;  1321          3040  2      !--
;  1322          3041  2      STORAGE_DESC = MODULE_TO_ADD [MC_NT_STORAGE];
;  1323          3042  3      IF (.STORAGE_DESC [STOR_BEGIN_RST] NEQ 0)
;  1324          3043  2      THEN
;  1325          3044  2              RETURN(FALSE);
;  1326          3045  2
;  1327          3046  2      !++
;  1328          3047  2      ! If we are going to allocate stroage for any SATs or LVTs, again check that
;  1329          3048  2      ! none is already allocated.                                                                              ;
;  1330          3049  2      !--
;  1331          3050  3      IF (.MODULE_TO_ADD [MC_STATICS] NEQ 0)
;  1332          3051  2      THEN
;  1333          3052  3              BEGIN
;  1334          3053  3              STORAGE_DESC = MODULE_TO_ADD [MC_SAT_STORAGE];
;  1335          3054  4              IF (.STORAGE_DESC [STOR_BEGIN_RST] NEQ 0)
;  1336          3055  3              THEN
;  1337          3056  3                      RETURN(FALSE);
;  1338          3057  2              END;
;  1339          3058  3      IF (.MODULE_TO_ADD [MC_LITERALS] NEQ 0)
;  1340          3059  2      THEN
;  1341          3060  3              BEGIN
;  1342          3061  3              STORAGE_DESC = MODULE_TO_ADD [MC_LVT_STORAGE];
;  1343          3062  4              IF (.STORAGE_DESC [STOR_BEGIN_RST] NEQ 0)
;  1344          3063  3              THEN
```

```
;  1345     3064  3                    RETURN(FALSE);
;  1346     3065  2            END;
;  1347     3066  2
;  1348     3067  2    !++
;  1349     3068  2    ! At this point there is more to do only if the given MC pointer is to the
;  1350     3069  2    ! 'global' MC record.  This is  because there is no DST checking or language
;  1351     3070  2    ! verification for the global MC case.
;  1352     3071  2    !--
;  1353     3072  3    IF (.LANGUAGE_ADDR NEQ 0)
;  1354     3073  2    THEN
;  1355     3074  3            BEGIN
;  1356     3075  3            !++
;  1357     3076  3            ! Processing for all DST-originated modules.
;  1358     3077  3            !
;  1359     3078  3            ! First, pick up the supposed DST record ID of where the DST for this
;  1360     3079  3            ! module begins.
;  1361     3080  3            !--
;  1362     3081  3            BEGIN_DST_AT = .MODULE_TO_ADD [MC_DST_START];
;  1363     3082  3
;  1364     3083  3            !++
;  1365     3084  3            ! Position the DST to the indicated record, and
;  1366     3085  3            ! make sure that it is of type MODULE.
;  1367     3086  3            !--
;  1368     3087  4            IF ((DST_RECRD = PAT$POSITON_DST( .BEGIN_DST_AT )) EQL 0)
;  1369     3088  3            THEN
;  1370     3089  4                    BEGIN
;  1371     3090  4                    !++
;  1372     3091  4                    ! The supposed record does not exist.
;  1373     3092  4                    !--
;  1374     3093  4                    RETURN(FALSE);
;  1375     3094  4                    END
;  1376     3095  3            ELSE
;  1377     3096  3                    !++
;  1378     3097  3                    ! The record is reachable.  Make sure it is MODULE.
;  1379     3098  3                    !--
;  1380     3099  4                    IF (.DST_RECRD [DSTR_TYPE] NEQ DSC$K_DTYPE_MOD)
;  1381     3100  3                    THEN
;  1382     3101  4                            BEGIN
;  1383     3102  4                            $FAO_TT_OUT('!/DST MODULE misplaced.');
;  1384     3103  4                            RETURN(FALSE);
;  1385     3104  3                            END;
;  1386     3105  3
;  1387     3106  3            !++
;  1388     3107  3            ! Record type is OK.  Check that we know which source language the
;  1389     3108  3            ! module was written in.  If it's not BLISS or FORTRAN then treat it
;  1390     3109  3            ! as though it were MACRO.
;  1391     3110  3            !--
;  1392     3111  3            IF ((CURENT_LANGUAGE = .DST_RECRD[DSTR_VALUE]) NEQ BLISS_MODULE) AND
;  1393     3112  4               (.CURENT_LANGUAGE NEQ FORTRAN_MODULE)
;  1394     3113  3            THEN
;  1395     3114  3                    CURENT_LANGUAGE = MARS_MODULE ;
;  1396     3115  3
;  1397     3116  3            !++
;  1398     3117  3            ! Pass back the language code in which this module is written.  This is
;  1399     3118  3            ! needed because the caller of this routine doesn't get to look at the
;  1400     3119  3            ! DST record for this module.
;  1401     3120  3            !--
```

```
;  1402     3121  3          LANGUAGE_ADDR[0] = .CURENT_LANGUAGE;
;  1403     3122  2          END;
;  1404     3123  2
;  1405     3124  2      !++
;  1406     3125  2      ! Now that nothing can go wrong, go ahead and allocate all
;  1407     3126  2      ! NT, and possibly all SAT and/or all LVT storage.
;  1408     3127  2      !--
;  1409     3128  2      PAT$VS_INIT(    MODULE_TO_ADD [MC_NT_STORAGE],
;  1410     3129  2                      .MODULE_TO_ADD [MC_NAMES],
;  1411     3130  2                      RST_NT_SIZE
;  1412     3131  2                    );
;  1413     3132  2
;  1414     3133  2      !++
;  1415     3134  2      ! NT storage is accessed via RST-pointers.
;  1416     3135  2      !--
;  1417     3136  2      MODULE_TO_ADD [MC_NT_STORAGE] = FALSE;
;  1418     3137  2
;  1419     3138  2      !++
;  1420     3139  2      ! Likewise for SAT and LVT storage.  Only here we don't mind if no storage will
;  1421     3140  2      ! be needed.
;  1422     3141  2      !--
;  1423     3142  3      IF (.MODULE_TO_ADD [MC_STATICS] NEQ 0)
;  1424     3143  2      THEN
;  1425     3144  3          BEGIN
;  1426     3145  3          PAT$VS_INIT(    MODULE_TO_ADD [MC_SAT_STORAGE],
;  1427     3146  3                          .MODULE_TO_ADD [MC_STATICS],
;  1428     3147  3                          RST_SAT_SIZE
;  1429     3148  3                        );
;  1430     3149  3
;  1431     3150  3          !++
;  1432     3151  3          ! SAT storage is not accessed via RST-pointers.
;  1433     3152  3          !--
;  1434     3153  3          MODULE_TO_ADD [MC_SAT_STORAGE] = TRUE;
;  1435     3154  2          END;
;  1436     3155  3      IF (.MODULE_TO_ADD [MC_LITERALS] NEQ 0)
;  1437     3156  2      THEN
;  1438     3157  3          BEGIN
;  1439     3158  3          PAT$VS_INIT(    MODULE_TO_ADD [MC_LVT_STORAGE],
;  1440     3159  3                          .MODULE_TO_ADD [MC_LITERALS],
;  1441     3160  3                          RST_LVT_SIZE
;  1442     3161  3                        );
;  1443     3162  3
;  1444     3163  3          !++
;  1445     3164  3          ! LVT storage is not accessed via RST-pointers.
;  1446     3165  3          !--
;  1447     3166  3          MODULE_TO_ADD [MC_LVT_STORAGE] = TRUE;
;  1448     3167  2          END;
;  1449     3168  2
;  1450     3169  2      !++
;  1451     3170  2      ! Set the OWN variable which various other routines in this module work from,
;  1452     3171  2      ! to indicate which module we are currently building.  This is the only place
;  1453     3172  2      ! where this variable gets set.
;  1454     3173  2      !--
;  1455     3174  2      CURRENT_MODULE = .MODULE_TO_ADD;
;  1456     3175  2
;  1457     3176  2      !++
;  1458     3177  2      ! Looks like we're OK to go ahead and try to! add the indicated module.  This is
```

```
; 1459          3178  2 ! the only place in this routine where we return TRUE status.
; 1460          3179  2 !--
; 1461          3180  2 RETURN(TRUE);
; 1462          3181  1 END;
; INFO#212                 L1:3102
; Null expression appears in value-required context


                                                                        .PSECT  _PAT$PLIT,NOWRT,NOEXE,0

                                                                17  00118 P.AAJ:  .BYTE   23
69  6D  20  45  4C  55  44  4F  4D  20  54  53  44  2F  21  00119          .ASCII  \!/DST MODULE misplaced.\                        ;
                        2E  64  65  63  61  6C  70  73  00128                                                                   ;


                                                                        .PSECT  _PAT$CODE,NOWRT,2

                                                        003C 00000 OK_TO_ADD:
                                                                        .WORD   Save R2,R3,R4,R5                                ; 2950
                                55 00000000G    EF  9E 00002            MOVAB   PAT$VS_INIT, R5                                 ;
                                54 00000000G    EF  9E 00009            MOVAB   PAT$GL_RST_BEGN, R4                             ;
                    00000000G   EF              00  FB 00010            CALLS   #0, PAT$REPORT_FREE                             ; 3022
                                52              50  D0 00017            MOVL    R0, R2                                          ;
                                53      04      AC  D0 0001A            MOVL    MODULE_TO_ADD, R3                               ;
                                                53  DD 0001E            PUSHL   R3                                              ;
                    00000000G   EF              01  FB 00020            CALLS   #1, PAT$MODULE_SIZE                             ;
                                50      0BB8    C0  9E 00027            MOVAB   3000(R0), R0                                    ;
                                50              52  D1 0002C            CMPL    R2, R0                                          ;
                                                5B  15 0002F            BLEQ    3$                                              ;
          51                    53              64  C1 00031            ADDL3   PAT$GL_RST_BEGN, R3, R1                         ; 3034
                                            08  A1  D5 00035            TSTL    8(R1)                                           ;
                                                52  13 00038            BEQL    3$                                              ;
                                50          1C  A1  9E 0003A            MOVAB   28(R1), STORAGE_DESC                            ; 3041
                                            01  A0  B5 0003E            TSTW    1(STORAGE_DESC)                                 ; 3042
                                                49  12 00041            BNEQ    3$                                              ;
                                            31  A1  D5 00043            TSTL    49(R1)                                          ; 3050
                                                09  13 00046            BEQL    1$                                              ;
                                50          23  A1  9E 00048            MOVAB   35(R1), STORAGE_DESC                            ; 3053
                                            01  A0  B5 0004C            TSTW    1(STORAGE_DESC)                                 ; 3054
                                                3B  12 0004F            BNEQ    3$                                              ;
                                            35  A1  D5 00051 1$:        TSTL    53(R1)                                          ; 3058
                                                09  13 00054            BEQL    2$                                              ;
                                50          2A  A1  9E 00056            MOVAB   42(R1), STORAGE_DESC                            ; 3061
                                            01  A0  B5 0005A            TSTW    1(STORAGE_DESC)                                 ; 3062
                                                2D  12 0005D            BNEQ    3$                                              ;
                                            08  AC  D5 0005F 2$:        TSTL    LANGUAGE_ADDR                                   ; 3072
                                                3E  13 00062            BEQL    6$                                              ;
                                50          04  A1  D0 00064            MOVL    4(R1), BEGIN_DST_AT                             ; 3081
                                                50  DD 00068            PUSHL   BEGIN_DST_AT                                    ; 3087
                    00000000G   EF              01  FB 0006A            CALLS   #1, PAT$POSITON_DST                             ;
                                52              50  D0 00071            MOVL    R0, DST_RECRD                                   ;
                                                16  13 00074            BEQL    3$                                              ;
                          BC    8F          01  A2  91 00076            CMPB    1(DST_RECRD), #188                              ; 3099
                                                11  13 0007B            BEQL    4$                                              ;
                                                7E  D4 0007D            CLRL    -(SP)                                           ; 3102
```

```
                00000000'  EF  9F 0007F          PUSHAB  P.AAJ                                   ;
          00000000G    EF      02 FB 00085          CALLS   #2, PAT$FAO_OUT                       ;
                               69 11 0008C 3$:      BRB     9$                                    ; 3103
                     50    03  A2 D0 0008E 4$:      MOVL    3(DST_RECRD), CURENT_LANGUAGE         ; 3111
                     02        50 D1 00092          CMPL    CURENT_LANGUAGE, #2                   ;
                               07 13 00095          BEQL    5$                                    ;
                     01        50 D1 00097          CMPL    CURENT_LANGUAGE, #1                   ; 3112
                               02 13 0009A          BEQL    5$                                    ;
                               50 D4 0009C          CLRL    CURENT_LANGUAGE                       ; 3114
                08   BC        50 D0 0009E 5$:      MOVL    CURENT_LANGUAGE, @LANGUAGE_ADDR       ; 3121
                               2C DD 000A2 6$:      PUSHL   #44                                   ; 3128
              50       53      64 C1 000A4          ADDL3   PAT$GL_RST_BEGN, R3, R0               ; 3129
                          08   A0 DD 000A8          PUSHL   8(R0)                                 ;
                          1C   A0 9F 000AB          PUSHAB  28(R0)                                ; 3128
                     65        03 FB 000AE          CALLS   #3, PAT$VS_INIT                       ;
              50       53      64 C1 000B1          ADDL3   PAT$GL_RST_BEGN, R3, R0               ; 3136
                          1C   A0 94 000B5          CLRB    28(R0)                                ;
                          31   A0 D5 000B8          TSTL    49(R0)                                ; 3142
                               13 13 000BB          BEQL    7$                                    ;
                               0A DD 000BD          PUSHL   #10                                   ; 3145
                          31   A0 DD 000BF          PUSHL   49(R0)                                ; 3146
                          23   A0 9F 000C2          PUSHAB  35(R0)                                ; 3145
                     65        03 FB 000C5          CALLS   #3, PAT$VS_INIT                       ;
              50       53      64 C1 000C8          ADDL3   PAT$GL_RST_BEGN, R3, R0               ; 3153
                    23   A0      01 90 000CC          MOVB    #1, 35(R0)                            ;
              50       53      64 C1 000D0 7$:      ADDL3   PAT$GL_RST_BEGN, R3, R0               ; 3155
                          35   A0 D5 000D4          TSTL    53(R0)                                ;
                               13 13 000D7          BEQL    8$                                    ;
                               06 DD 000D9          PUSHL   #6                                    ; 3158
                          35   A0 DD 000DB          PUSHL   53(R0)                                ; 3159
                          2A   A0 9F 000DE          PUSHAB  42(R0)                                ; 3158
                     65        03 FB 000E1          CALLS   #3, PAT$VS_INIT                       ;
              50       53      64 C1 000E4          ADDL3   PAT$GL_RST_BEGN, R3, R0               ; 3166
                    2A   A0      01 90 000E8          MOVB    #1, 42(R0)                            ;
          00000000'    EF      53 D0 000EC 8$:      MOVL    R3, CURRENT_MODULE                    ; 3174
                     50        01 D0 000F3          MOVL    #1, R0                                ; 3180
                               04 000F6             RET                                           ;
                               50 D4 000F7 9$:      CLRL    R0                                    ; 3181
                                  04 000F9          RET                                           ;

; Routine Size:  250 bytes,    Routine Base:  _PAT$CODE + 04A8
```

```
;  1464          3182  1 ROUTINE ADD_NT( DST_RECRD, UP_SCOPE ) =
;  1465          3183  1
;  1466          3184  1 !++
;  1467          3185  1 ! Functional Description:
;  1468          3186  1 !
;  1469          3187  1 !       Add a symbol to the name table (NT).
;  1470          3188  1 !
;  1471          3189  1 ! Formal Parameters:
;  1472          3190  1 !       DST_RECRD       -a pointer to the DST record that corresponds
;  1473          3191  1 !                        to the symbol being added.
;  1474          3192  1 !       UP_SCOPE        -an RST-pointer to the NT or MC record that
;  1475          3193  1 !                        corresponds to the symbol that is scope-wise
;  1476          3194  1 !                        above the symbol we are to add.
;  1477          3195  1 !
;  1478          3196  1 ! Implicit Inputs:
;  1479          3197  1 !
;  1480          3198  1 !       The OWN, CURRENT_MODULE, has been set up to
;  1481          3199  1 !       indicate the MC pointer to the module we are
;  1482          3200  1 !       currently adding.
;  1483          3201  1 !
;  1484          3202  1 !       We call PAT$DST_VALUE with a DST_RECRD pointer - NOT a
;  1485          3203  1 !       so-called DST_REC_ID.  This works, for now, because these
;  1486          3204  1 !       two concepts are still the same.  If/when this is no longer
;  1487          3205  1 !       true, we must change it and all calls to it so that only one
;  1488          3206  1 !       way of specifying a DST record is used.
;  1489          3207  1 !
;  1490          3208  1 ! Implicit Outputs:
;  1491          3209  1 !
;  1492          3210  1 !       None.
;  1493          3211  1 !
;  1494          3212  1 ! Routine Value:
;  1495          3213  1 !
;  1496          3214  1 !       FALSE, if some real error occurs,
;  1497          3215  1 !       an NT_PTR to where we did add or find it, otherwise.
;  1498          3216  1 !
;  1499          3217  1 ! Side Effects:
;  1500          3218  1 !
;  1501          3219  1 !       The NT gets updated to reflect the addition of the new
;  1502          3220  1 !       symbol if 1) PAT$DST_VALUE knows how to evaluate the
;  1503          3221  1 !       given DST, and 2) We have not already built an NT
;  1504          3222  1 !       record for this symbol (because it also came in as a GSD).
;  1505          3223  1 !--
;  1506          3224  1
;  1507          3225  2 BEGIN
;  1508          3226  2
;  1509          3227  2 LABEL
;  1510          3228  2         NEXT_DUP_LOOP;
;  1511          3229  2
;  1512          3230  2 MAP
;  1513          3231  2         UP_SCOPE : REF MC_RECORD,                    ! This is an example of a case where we woul
;  1514          3232  2                                                      ! in trouble if pointers into the MC were no
;  1515          3233  2                                                      ! same as those into the NT because here we
;  1516          3234  2                                                      ! don't yet know to which structure UP_SCOPE
;  1517          3235  2         DST_RECRD : REF DST_RECORD;                  ! Pointer to the given DST record
;  1518          3236  2                                                      ! which we use only for 'standard' DST
;  1519          3237  2                                                      ! structures.
;  1520          3238  2
```

```
;  1521          3239  2 LOCAL
;  1522          3240  2         LOOKUP_STATUS,                          ! SUCCESS CODE FROM PAT$DST_VALUE
;  1523          3241  2         NT_HASH : REF NT_RECORD,                ! Hash index used to access NT chains.
;  1524          3242  2         NEW_NT_PTR : REF NT_RECORD,             ! Pointer to NT record where the new
;  1525          3243  2                                                 ! symbol gets inserted.
;  1526          3244  2         NT_PTR : REF NT_RECORD,                 ! NT pointer used to scan along chains.
;  1527          3245  2         DST_NAME_CS : CS_POINTER,               ! Pointer to where the symbol name CS
;  1528          3246  2                                                 ! is in the DST record.
;  1529          3247  2         SYMBL_VALUE;                            ! The value of the symbol we are to add.
;  1530          3248  2
;  1531          3249  2 !++
;  1532          3250  2 ! There is no point in adding an NT record for a symbol
;  1533          3251  2 ! we can't evaluate.
;  1534          3252  2 !--
;  1535          3253  3 IF ((LOOKUP_STATUS =  PAT$DST_VALUE( .DST_RECRD, SYMBL_VALUE)) EQL 0)   ! ++038
;  1536          3254  2 THEN
;  1537          3255  3         BEGIN
;  1538          3256  3         RETURN(FALSE);
;  1539          3257  2         END;
;  1540          3258  2
;  1541          3259  2 !++
;  1542          3260  2 ! The only difference in building an NT record from either of the three classes
;  1543          3261  2 ! of DST records we distinguish, (type zero, SRM types, and 'the rest'),
;  1544          3262  2 ! is the calculation of where in the record the name counted string is found.
;  1545          3263  2 !
;  1546          3264  2 ! This code relies on the fact that the TYPE fields are in the same place in
;  1547          3265  2 ! the three types of DST records.
;  1548          3266  2 !--
;  1549          3267  2 DST_NAME_CS = DST_RECRD [DSTR_NAME];
;  1550          3268  2
;  1551          3269  2 !++
;  1552          3270  2 ! Add the symbol to the name table.  Now we simply determine whether to add it
;  1553          3271  2 ! onto the end of an existing hash chain, or whether to begin a new one.
;  1554          3272  2 !--
;  1555          3273  2 NT_HASH = PAT$NT_HASH_FCN( .DST_NAME_CS );
;  1556          3274  3 IF ((NT_PTR = .PAT$GL_NT_HASH [.NT_HASH]) NEQ 0)
;  1557          3275  2 THEN
;  1558          3276  3         BEGIN
;  1559          3277  3         !++
;  1560          3278  3         ! There are already some symbols that hash to this same value.  This
;  1561          3279  3         ! means that we simply link the new  record onto the end of this chain.
;  1562          3280  3         !--
;  1563          3281  4         WHILE( .NT_PTR [NT_FORWARD] NEQ 0 )
;  1564          3282  3         DO
;  1565          3283  3                 !++
;  1566          3284  3                 ! Skip along to the end of the chain.
;  1567          3285  3                 !--
;  1568          3286  3                 NT_PTR = .NT_PTR [NT_FORWARD];
;  1569          3287  3         END
;  1570          3288  2 ELSE
;  1571          3289  3         BEGIN
;  1572          3290  3         !++
;  1573          3291  3         ! To build a new chain, we only need to know where in the hash vector
;  1574          3292  3         ! to link it on. Make this hash-vector address look like a pointer
;  1575          3293  3         ! to an NT record so that we can fill in NT_FORWARD below thereby filling
;  1576          3294  3         ! in the hash vector.
;  1577          3295  3         !--
```

```
;  1578          3296  3          NT_PTR = PAT$GL_NT_HASH [.NT_HASH] - .PAT$GL_RST_BEGN;
;  1579          3297  2          END;
;  1580          3298  2
;  1581          3299  2 !++
;  1582          3300  2 ! Allocate space for the new NT record and fill in the fixed fields.  This must
;  1583          3301  2 ! go without hitch or we give up.  Note that the NT record we build is the same
;  1584          3302  2 ! no matter what type of DST record we build it from.
;  1585          3303  2 !--
;  1586          3304  4 IF ((NEW_NT_PTR = PAT$VS_GET( CURRENT_MODULE [MC_NT_STORAGE], RST_NT_OVERHEAD + .DST_NAME_CS[0]))
;  1587          3305  3    EQL 0)
;  1588          3306  2 THEN
;  1589          3307  2          !++
;  1590          3308  2          ! We ran out of space.  This should never happen because we supposedly
;  1591          3309  2          ! knew how much vector storage to allocate before we began adding any NTs.
;  1592          3310  2          !--
;  1593          3311  2          RETURN(FALSE);
;  1594          3312  2
;  1595          3313  2 NEW_NT_PTR [NT_UP_SCOPE] = .UP_SCOPE;
;  1596          3314  2 NEW_NT_PTR [NT_DST_PTR] = .DST_RECRD;
;  1597          3315  2 NEW_NT_PTR [NT_TYPE] = .DST_RECRD [DSTR_TYPE];
;  1598          3316  2
;  1599          3317  2 !++
;  1600          3318  2 ! Moving the name could cause problems if we change the data representation.
;  1601          3319  2 !  Note that here we move both the name-size count as well as the name itself.
;  1602          3320  2 !--
;  1603          3321  2 CH$MOVE( .DST_NAME_CS[0] +1, .DST_NAME_CS, NEW_NT_PTR [NT_NAME_CS] );
;  1604          3322  2
;  1605          3323  2 !++
;  1606          3324  2 ! Add this new NT record to the name table by connecting up the hash-chain links.
;  1607          3325  2 !  The new record points back to a previous NT record, OR, it points
;  1608          3326  2 ! back to the hash vector itself.  In either case there  is no NT record FORWARD
;  1609          3327  2 ! of the new record.
;  1610          3328  2 !--
;  1611          3329  2 NEW_NT_PTR [NT_BACKWARD] = .NT_PTR;
;  1612          3330  2 NEW_NT_PTR [NT_FORWARD] = 0;
;  1613          3331  2
;  1614          3332  2 !++
;  1615          3333  2 ! The following writes into the NT_FORWARD field of a previous NT record, OR,
;  1616          3334  2 ! it writes the first chain-pointer into the hash chain itself.  This
;  1617          3335  2 ! is why NT_FORWARD has to be the first field of the NT_FIELD_SET.
;  1618          3336  2 !--
;  1619          3337  2 NT_PTR [NT_FORWARD] = .NEW_NT_PTR;
;  1620          3338  2
;  1621          3339  2 !++
;  1622          3340  2 ! All went ok so we return an RST-pointer to where this symbol got added.
;  1623          3341  2 !--
;  1624          3342  2 RETURN(.NEW_NT_PTR);
;  1625          3343  1 END;


                                  03FC 00000 ADD_NT:  .WORD   Save R2,R3,R4,R5,R6,R7,R8,R9                    ; 3182
                59 00000000G   EF   9E 00002          MOVAB   PAT$GL_RST_BEGN, R9                             ;
                5E             04   C2 00009          SUBL2   #4, SP                                          ;
                               5E   DD 0000C          PUSHL   SP                                              ; 3253
```

```
                        53        04    AC  D0 0000F          MOVL    DST_RECRD, R3                                   ;
                                        53  DD 00012          PUSHL   R3                                              ;
                    00000000G   EF          02  FB 00014      CALLS   #2, PAT$DST_VALUE                               ;
                                            50  D5 0001B      TSTL    LOOKUP_STATUS                                   ;
                                            74  13 0001D      BEQL    4$                                              ;
                        52        07    A3  9E 0001F          MOVAB   7(R3), DST_NAME_CS                              ; 3267
                                        52  DD 00023          PUSHL   DST_NAME_CS                                     ; 3273
                    00000000G   EF          01  FB 00025      CALLS   #1, PAT$NT_HASH_FCN                             ;
                                50 00000000GFF40  3E 0002C      MOVAW   @PAT$GL_NT_HASH[NT_HASH], R0                    ; 3274
                                56          60  3C 00034      MOVZWL  (R0), NT_PTR                                    ;
                                            0D  13 00037      BEQL    2$                                              ;
                50                      56  69  C1 00039  1$: ADDL3   PAT$GL_RST_BEGN, NT_PTR, R0                    ; 3281
                                            60  B5 0003D      TSTW    (R0)                                            ;
                                            09  13 0003F      BEQL    3$                                              ;
                                        56  60  3C 00041      MOVZWL  (R0), NT_PTR                                    ; 3286
                                            F3  11 00044      BRB     1$                                              ;
                56                      50  69  C3 00046  2$: SUBL3   PAT$GL_RST_BEGN, R0, NT_PTR                    ; 3296
                                        7E  62  9A 0004A  3$: MOVZBL  (DST_NAME_CS), -(SP)                            ; 3304
                                        6E  0D  C0 0004D      ADDL2   #13, (SP)                                       ;
                50 00000000'    EF          69  C1 00050      ADDL3   PAT$GL_RST_BEGN, CURRENT_MODULE, R0              ;
                                    1C  A0  9F 00058          PUSHAB  28(R0)                                          ;
                    00000000G   EF          02  FB 0005B      CALLS   #2, PAT$VS_GET                                  ;
                                58          50  D0 00062      MOVL    R0, NEW_NT_PTR                                  ;
                                            2C  13 00065      BEQL    4$                                              ; 3305
                57                      58  69  C1 00067      ADDL3   PAT$GL_RST_BEGN, NEW_NT_PTR, R7                 ; 3313
                            08  A7      08  AC  B0 0006B      MOVW    UP_SCOPE, 8(R7)                                 ;
                            04  A7          53  D0 00070      MOVL    R3, 4(R7)                                       ; 3314
                            02  A7      01  A3  90 00074      MOVB    1(R3), 2(R7)                                    ; 3315
                                50          62  9A 00079      MOVZBL  (DST_NAME_CS), R0                               ; 3321
                                            50  D6 0007C      INCL    R0                                              ;
            0C  A7                  62          50  28 0007E  MOVC3   R0, (DST_NAME_CS), 12(R7)                       ;
                            0A  A7          56  B0 00083      MOVW    NT_PTR, 10(R7)                                  ; 3329
                                            67  B4 00087      CLRW    (R7)                                            ; 3330
                                56          69  C0 00089      ADDL2   PAT$GL_RST_BEGN, R6                             ; 3337
                                66          58  B0 0008C      MOVW    NEW_NT_PTR, (R6)                                ;
                                50          58  D0 0008F      MOVL    NEW_NT_PTR, R0                                  ; 3342
                                            04  00092      RET                                                     ;
                                            50  D4 00093  4$: CLRL    R0                                              ; 3343
                                                04 00095      RET                                                     ;

; Routine Size:  150 bytes,    Routine Base:  _PAT$CODE + 05A2
```

```
;  1627     3344  1 ROUTINE ADD_SAT( NT_PTR, LOWER, UPPER ) =
;  1628     3345  1
;  1629     3346  1 !++
;  1630     3347  1 ! Functional Description:
;  1631     3348  1 !
;  1632     3349  1 !       Build the SAT record to correspond to an NT record.
;  1633     3350  1 !
;  1634     3351  1 ! Formal Parameters:
;  1635     3352  1 !       NT_PTR          -An RST-pointer to the NT record for this symbol.
;  1636     3353  1 !       LOWER           -The lower-bound value which is bound to
;  1637     3354  1 !                        this symbol.
;  1638     3355  1 !       UPPER           -The upper-bound value which is bound to
;  1639     3356  1 !                        this symbol.
;  1640     3357  1 !
;  1641     3358  1 ! Implicit Inputs:
;  1642     3359  1 !
;  1643     3360  1 !       Symbols marked NT_IS_GLOBAL have already been added
;  1644     3361  1 !       to the SAT so we don't do it again.  We do update
;  1645     3362  1 !       SAT information therein, though, once we have found
;  1646     3363  1 !       out where we added it the last time.
;  1647     3364  1 !
;  1648     3365  1 !       The OWN, CURRENT_MODULE, has been set up to
;  1649     3366  1 !       indicate the MC pointer to the module we are
;  1650     3367  1 !       currently adding.  Storage allocation works
;  1651     3368  1 !       from the vector storage descriptors stored therein.
;  1652     3369  1 !
;  1653     3370  1 ! Implicit Outputs:
;  1654     3371  1 !
;  1655     3372  1 !       None.
;  1656     3373  1 !
;  1657     3374  1 ! Routine Value:
;  1658     3375  1 !
;  1659     3376  1 !       0, if the SAT record was not built,
;  1660     3377  1 !       a SAT-pointer to where it was build, otherwise.
;  1661     3378  1 !
;  1662     3379  1 ! Side Effects:
;  1663     3380  1 !
;  1664     3381  1 !       The SAT gets updated to reflect the addition of the new
;  1665     3382  1 !       symbol.
;  1666     3383  1 !
;  1667     3384  1 !       The NT bit, NT_IS_BOUNDED is set to TRUE or FALSE,
;  1668     3385  1 !       depending on whether the given UPPER_BOUND is 0 or not.
;  1669     3386  1 !--
;  1670     3387  1
;  1671     3388  2 BEGIN
;  1672     3389  2
;  1673     3390  2 MAP
;  1674     3391  2         NT_PTR : REF NT_RECORD;
;  1675     3392  2
;  1676     3393  2 LOCAL
;  1677     3394  2         SAT_PTR : REF SAT_RECORD;
;  1678     3395  2
;  1679     3396  2 SAT_PTR = 0;
;  1680     3397  2
;  1681     3398  2 !++
;  1682     3399  2 ! Ask for space for the fixed-size record. This should not fail because
;  1683     3400  2 ! we supposedly knew how much space we wanted and allocated it all
```

```
;  1684          3401  2 ! before we got here.
;  1685          3402  2 !--
;  1686          3403  3 IF ((SAT_PTR = PAT$VS_GET(CURRENT_MODULE [MC_SAT_STORAGE], RST_SAT_SIZE)) EQL 0)
;  1687          3404  2 THEN
;  1688          3405  3         BEGIN
;  1689          3406  3         $FAO_TT_OUT('!/sat overflow');                          ! Don't allow the SAT to overflow.
;  1690          3407  3         RETURN(FALSE);
;  1691          3408  2         END;
;  1692          3409  2
;  1693          3410  2 !++
;  1694          3411  2 ! Otherwise go ahead and fill in the new record or update the old one.
;  1695          3412  2 !--
;  1696          3413  2 SAT_PTR [SAT_LB] = .LOWER;
;  1697          3414  2 SAT_PTR [SAT_UB] = .UPPER;
;  1698          3415  2 SAT_PTR [SAT_NT_PTR] = .NT_PTR;
;  1699          3416  2
;  1700          3417  2 !++
;  1701          3418  2 ! Mark all NTs _BOUNDED or not depending on whether the (new) upper bound is 0
;  1702          3419  2 ! or not.
;  1703          3420  2 !--
;  1704          3421  3 IF (.UPPER NEQ 0)
;  1705          3422  2 THEN
;  1706          3423  2         NT_PTR [NT_IS_BOUNDED] = TRUE;
;  1707          3424  2
;  1708          3425  2 !++
;  1709          3426  2 ! Return a pointer to where the SAT record was built.  Usually this is only
;  1710          3427  2 ! tested for non-zero, PAT$ADD_MODULE keeps a stack of SAT record pointers
;  1711          3428  2 ! for ROUTINEs, though, so it can go back and put in the UB value since we
;  1712          3429  2 ! don't get that until end-of-routine.
;  1713          3430  2 !--
;  1714          3431  2 RETURN(.SAT_PTR);
;  1715          3432  1 END;
;  INFO#212                L1:3406
;  Null expression appears in value-required context


                                                                .PSECT  _PAT$PLIT,NOWRT,NOEXE,0

                                                        0E  00130 P.AAK:  .BYTE   14                                              ;
   77  6F  6C  66  72  65  76  6F  20  74  61  73  2F   21  00131        .ASCII  \!/sat overflow\                                      ;


                                                                .PSECT  _PAT$CODE,NOWRT,2

                                                      000C  00000 ADD_SAT:.WORD   Save R2,R3                                      ; 3344
                                          53 00000000G  EF   9E  00002        MOVAB   PAT$GL_RST_BEGN, R3                           ;
                                                        52   D4  00009        CLRL    SAT_PTR                                       ; 3396
                                                        0A   DD  0000B        PUSHL   #10                                           ; 3403
                                50 00000000'  EF              63   C1  0000D        ADDL3   PAT$GL_RST_BEGN, CURRENT_MODULE, R0   ;
                                                        23    A0   9F  00015        PUSHAB  35(R0)                                ;
                                   00000000G  EF              02   FB  00018        CALLS   #2, PAT$VS_GET                        ;
                                              52              50   D0  0001F        MOVL    R0, SAT_PTR                           ;
                                                              11   12  00022        BNEQ    1$                                    ;
                                                              7E   D4  00024        CLRL    -(SP)                                 ; 3406
                                                   00000000'  EF   9F  00026        PUSHAB  P.AAK                                 ;
```

```
                 00000000G  EF              02  FB 0002C          CALLS   #2, PAT$FAO_OUT                          ;
                                            1C  11 00033          BRB     3$                                      ; 3407
                       02   A2        08    AC  7D 00035 1$:      MOVQ    LOWER, 2(SAT_PTR)                       ; 3413
                       62   04        AC    B0 0003A              MOVW    NT_PTR, (SAT_PTR)                       ; 3415
                            0C        AC    D5 0003E              TSTL    UPPER                                   ; 3421
                                            0A  13 00041          BEQL    2$                                      ;
                 50         04        AC    63  C1 00043          ADDL3   PAT$GL_RST_BEGN, NT_PTR, R0             ; 3423
                       03   A0        80    8F  88 00048          BISB2   #128, 3(R0)                             ;
                            50              52  D0 0004D 2$:      MOVL    SAT_PTR, R0                             ; 3431
                                            04  00050             RET                                             ;
                                            50  D4 00051 3$:      CLRL    R0                                      ; 3432
                                                04 00053          RET                                             ;

; Routine Size:  84 bytes,    Routine Base:  _PAT$CODE + 0638
```

```
;  1717          3433  1 ROUTINE ADD_LVT( NT_PTR, LVT_VAL ) =
;  1718          3434  1
;  1719          3435  1 !++
;  1720          3436  1 ! Functional Description:
;  1721          3437  1 !
;  1722          3438  1 !       Build the LVT record to correspond to the
;  1723          3439  1 !       symbol we have just built the indicated NT
;  1724          3440  1 !       record for.
;  1725          3441  1 !
;  1726          3442  1 ! Formal Parameters:
;  1727          3443  1 !
;  1728          3444  1 !       NT_PTR          -An RST-pointer to the NT record for this symbol.
;  1729          3445  1 !       LVT_VAL         -The literal value which is bound to
;  1730          3446  1 !                        this symbol.
;  1731          3447  1 !
;  1732          3448  1 ! Implicit Inputs:
;  1733          3449  1 !
;  1734          3450  1 !       The OWN, CURRENT_MODULE, has been set up to
;  1735          3451  1 !       indicate the MC pointer to the module we are
;  1736          3452  1 !       currently adding.  Storage allocation works
;  1737          3453  1 !       from the vector storage descriptors stored therein.
;  1738          3454  1 !
;  1739          3455  1 ! Implicit Outputs:
;  1740          3456  1 !
;  1741          3457  1 !       None.
;  1742          3458  1 !
;  1743          3459  1 ! Routine Value:
;  1744          3460  1 !
;  1745          3461  1 !       TRUE, if all goes OK,
;  1746          3462  1 !       FALSE, otherwise.
;  1747          3463  1 !
;  1748          3464  1 ! Side Effects:
;  1749          3465  1 !
;  1750          3466  1 !       The LVT gets updated to reflect the addition of the new
;  1751          3467  1 !       symbol.
;  1752          3468  1 !--
;  1753          3469  1
;  1754          3470  2 BEGIN
;  1755          3471  2
;  1756          3472  2 MAP
;  1757          3473  2         NT_PTR : REF NT_RECORD;
;  1758          3474  2
;  1759          3475  2 LOCAL
;  1760          3476  2         LVT_PTR : REF LVT_RECORD;
;  1761          3477  2
;  1762          3478  2 !++
;  1763          3479  2 ! Ask for space for the fixed-size record. This should not fail because we
;  1764          3480  2 ! supposedly knew how much space we wanted and allocated it all before
;  1765          3481  2 ! we got here.
;  1766          3482  2 !--
;  1767          3483  4 IF ((LVT_PTR =  PAT$VS_GET(CURRENT_MODULE[MC_LVT_STORAGE], RST_LVT_SIZE))
;  1768          3484  3    EQL 0)
;  1769          3485  2 THEN
;  1770          3486  3         BEGIN
;  1771          3487  3         $FAO_TT_OUT('!/lvt overflow');                  ! Don't allow the LVT to overflow.
;  1772          3488  3         RETURN(FALSE);
;  1773          3489  2         END;
```

```
;  1774          3490  2
;  1775          3491  2 !++
;  1776          3492  2 ! Otherwise go ahead and fill in the new record.
;  1777          3493  2 !--
;  1778          3494  2 LVT_PTR [LVT_VALUE] = .LVT_VAL;
;  1779          3495  2 LVT_PTR [LVT_NT_PTR] = .NT_PTR;
;  1780          3496  2
;  1781          3497  2 RETURN(TRUE);
;  1782          3498  1 END;
;  INFO#212                L1:3487
;  Null expression appears in value-required context


                                                                .PSECT  _PAT$PLIT,NOWRT,NOEXE,0

                                                        0E  0013F P.AAL:  .BYTE   14                                              ;
   77  6F  6C  66  72  65  76  6F  20  74  76  6C  2F  21  00140         .ASCII  \!/lvt overflow\                                      ;


                                                                .PSECT  _PAT$CODE,NOWRT,2

                                                      0004  00000 ADD_LVT:.WORD   Save R2                                         ; 3433
                                                  06  DD  00002         PUSHL   #6                                              ; 3483
                      50 00000000'  EF 00000000G  EF  C1  00004         ADDL3   PAT$GL_RST_BEGN, CURRENT_MODULE, R0              ;
                                          2A      A0  9F  00010         PUSHAB  42(R0)                                          ;
                         00000000G  EF            02  FB  00013         CALLS   #2, PAT$VS_GET                                  ;
                                    52            50  D0  0001A         MOVL    R0, LVT_PTR                                     ;
                                                  11  12  0001D         BNEQ    1$                                              ; 3484
                                                  7E  D4  0001F         CLRL    -(SP)                                           ; 3487
                                    00000000'     EF  9F  00021         PUSHAB  P.AAL                                           ;
                                 00000000G  EF            02  FB  00027         CALLS   #2, PAT$FAO_OUT                         ;
                                                  0D  11  0002E         BRB     2$                                              ; 3488
                               02   A2      08   AC  D0  00030 1$:     MOVL    LVT_VAL, 2(LVT_PTR)                             ; 3494
                                    62      04   AC  B0  00035         MOVW    NT_PTR, (LVT_PTR)                               ; 3495
                                    50            01  D0  00039         MOVL    #1, R0                                          ; 3497
                                                      04  0003C         RET                                                     ;
                                                  50  D4  0003D 2$:     CLRL    R0                                              ; 3498
                                                      04  0003F         RET                                                     ;

; Routine Size:  64 bytes,    Routine Base:  _PAT$CODE + 068C
```

```
;1784        3499  1 GLOBAL ROUTINE PAT$GET_BOUNDS( DESCRIP_BLOCK, BOUNDS_DESC ) : NOVALUE =
;1785        3500  1
;1786        3501  1 !++
;1787        3502  1 ! Functional Description:
;1788        3503  1 !
;1789        3504  1 !       Deduce the virtual address of the beginning, and the length (in bytes),
;1790        3505  1 !       of the indicated array.  i.e. Given the SRM-defined 'array descriptor',
;1791        3506  1 !       produce the corresponding PATCH/RST-defined 'array bounds descriptor'.
;1792        3507  1 !       The latter is defined in PATRST.REQ
;1793        3508  1 !
;1794        3509  1 ! Formal Parameters:
;1795        3510  1 !
;1796        3511  1 !       DESCRIP_BLOCK   -address of SRM array descriptor
;1797        3512  1 !       BOUNDS_DESC     -address of the ARRAY_BNDS_DESC we are
;1798        3513  1 !                        to 'fill in'.
;1799        3514  1 !
;1800        3515  1 ! Implicit Inputs:
;1801        3516  1 !
;1802        3517  1 !       The SRM-defined notion of 'array descriptor'.
;1803        3518  1 !
;1804        3519  1 ! Implicit Outputs:
;1805        3520  1 !
;1806        3521  1 !       None.
;1807        3522  1 !
;1808        3523  1 ! Routine Value:
;1809        3524  1 !
;1810        3525  1 !       NOVALUE
;1811        3526  1 !
;1812        3527  1 ! Side Effects:
;1813        3528  1 !
;1814        3529  1 !       BOUNDS_DESC is an output parameter.  (see above)
;1815        3530  1 !--
;1816        3531  1
;1817        3532  2 BEGIN
;1818        3533  2
;1819        3534  2 MAP
;1820        3535  2         DESCRIP_BLOCK : REF BLOCK [,BYTE],
;1821        3536  2         BOUNDS_DESC : REF ARRAY_BNDS_DESC;
;1822        3537  2
;1823        3538  2 LOCAL
;1824        3539  2         SPAN_BLOCK : REF VECTOR,
;1825        3540  2         NUM_ARRAY_ELEMS;
;1826        3541  2
;1827        3542  2 !++
;1828        3543  2 ! Check that this is a valid array descriptor.  PATCH handles FORTRAN'S
;1829        3544  2 ! _CLASS_A (general array) and _CLASS_S (type CHARACTER).
;1830        3545  2 !--
;1831        3546  3 IF (.DESCRIP_BLOCK [DSC$B_CLASS] EQL DSC$K_CLASS_A)
;1832        3547  2 THEN
;1833        3548  3         BEGIN
;1834        3549  3         !++
;1835        3550  3         ! General array descriptors.  Check consistency and return if
;1836        3551  3         ! something is wrong.
;1837        3552  3         !--
;1838        3553  4         IF ((NOT .DESCRIP_BLOCK [DSC$V_FL_COLUMN]) OR
;1839        3554  4             (NOT .DESCRIP_BLOCK [DSC$V_FL_COEFF]) OR
;1840        3555  4             (NOT .DESCRIP_BLOCK [DSC$V_FL_BOUNDS]) OR
```

```
;  1841     3556  4            (.DESCRIP_BLOCK [DSC$B_DIMCT] LEQ 0))
;  1842     3557  3        THEN
;  1843     3558  4                BEGIN
;  1844     3559  4                SIGNAL( PAT$_INVARRDSC );
;  1845     3560  4                RETURN;
;  1846     3561  3                END;
;  1847     3562  3
;  1848     3563  3                !++
;  1849     3564  3                ! Calculate the required values for general arrays.  The length of the
;  1850     3565  3                ! array is simply tallied up by multiplying the total number of elements
;  1851     3566  3                ! by each element size.  The total number of elements is the product of
;  1852     3567  3                ! the number of elements in each dimension.
;  1853     3568  3                !--
;  1854     3569  3                SPAN_BLOCK = DESCRIP_BLOCK [DSC$L_M1];
;  1855     3570  3                NUM_ARRAY_ELEMS = .SPAN_BLOCK[0];
;  1856     3571  4                INCR COUNT FROM 1 TO (.DESCRIP_BLOCK [DSC$B_DIMCT] -1 )
;  1857     3572  3                DO
;  1858     3573  3                        NUM_ARRAY_ELEMS = .NUM_ARRAY_ELEMS * .SPAN_BLOCK[.COUNT];
;  1859     3574  3
;  1860     3575  3                !++
;  1861     3576  3                ! Calculate the length of the array, in bytes, and return this information.
;  1862     3577  3                !--
;  1863     3578  3                BOUNDS_DESC [ARRAY_LENGTH] = .NUM_ARRAY_ELEMS * .DESCRIP_BLOCK [DSC$W_LENGTH];
;  1864     3579  3
;  1865     3580  3                !++
;  1866     3581  3                ! End of special processing for _CLASS_A.
;  1867     3582  3                !--
;  1868     3583  3                END
;  1869     3584  2        ELSE
;  1870     3585  3                IF (.DESCRIP_BLOCK[DSC$B_CLASS] EQL DSC$K_CLASS_S)
;  1871     3586  2                THEN
;  1872     3587  3                        BEGIN
;  1873     3588  3                        !++
;  1874     3589  3                        ! Class S is for static strings.  Return the needed information.
;  1875     3590  3                        !--
;  1876     3591  3                        BOUNDS_DESC[ARRAY_LENGTH] = .DESCRIP_BLOCK[DSC$W_LENGTH];
;  1877     3592  3                        END
;  1878     3593  2                ELSE
;  1879     3594  3                        BEGIN
;  1880     3595  3                        !++
;  1881     3596  3                        ! No other classes currently supported.
;  1882     3597  3                        !--
;  1883     3598  3                        SIGNAL(PAT$_INVARRDSC);
;  1884     3599  3                        RETURN;
;  1885     3600  2                        END;
;  1886     3601  2
;  1887     3602  2        !++
;  1888     3603  2        ! Return the required information common to all types.  Currently, this is
;  1889     3604  2        ! only the address where the data begins.
;  1890     3605  2        !--
;  1891     3606  2        BOUNDS_DESC [ARRAY_ADDRESS] = .DESCRIP_BLOCK [DSC$A_POINTER];
;  1892     3607  2
;  1893     3608  1        END;
```

```
                                                001C 00000          .ENTRY  PAT$GET_BOUNDS, Save R2,R3,R4                   ; 3499
                                        52      04   AC   D0 00002          MOVL    DESCRIP_BLOCK, R2                       ; 3546
                                        04      03   A2   91 00006          CMPB    3(R2), #4                               ;
                                                     39   12 0000A          BNEQ    3$                                      ;
                                44              0A   A2   05 E1 0000C      BBC     #5, 10(R2), 4$                          ; 3553
                                3F              0A   A2   06 E1 00011      BBC     #6, 10(R2), 4$                          ; 3554
                                                0A   A2   95 00016          TSTB    10(R2)                                  ; 3555
                                                     3A   18 00019          BGEQ    4$                                      ;
                                                0B   A2   95 0001B          TSTB    11(R2)                                  ; 3556
                                                     35   13 0001E          BEQL    4$                                      ;
                                        50      14   A2   9E 00020          MOVAB   20(R2), SPAN_BLOCK                      ; 3569
                                        54           60   D0 00024          MOVL    (SPAN_BLOCK), NUM_ARRAY_ELEMS           ; 3570
                                        53      0B   A2   9A 00027          MOVZBL  11(R2), R3                              ; 3571
                                                     51   D4 0002B          CLRL    COUNT                                   ;
                                                     04   11 0002D          BRB     2$                                      ;
                                        54      6041 C4   0002F 1$:         MULL2   (SPAN_BLOCK)[COUNT], NUM_ARRAY_ELEMS    ; 3573
                                F8      51           53   F2 00033 2$:      AOBLSS  R3, COUNT, 1$                           ;
                                        51      08   AC   D0 00037          MOVL    BOUNDS_DESC, R1                         ; 3578
                                        50           62   3C 0003B          MOVZWL  (R2), R0                                ;
                            04  A1      54           50   C5 0003E          MULL3   R0, NUM_ARRAY_ELEMS, 4(R1)              ;
                                                     1E   11 00043          BRB     5$                                      ; 3546
                                        01      03   A2   91 00045 3$:      CMPB    3(R2), #1                               ; 3585
                                                     0A   12 00049          BNEQ    4$                                      ;
                                        50      08   AC   D0 0004B          MOVL    BOUNDS_DESC, R0                         ; 3591
                                            04  A0       62   3C 0004F      MOVZWL  (R2), 4(R0)                             ;
                                                     0E   11 00053          BRB     5$                                      ; 3585
                                            006D80CA 8F   DD 00055 4$:      PUSHL   #7176394                                ; 3598
                                00000000G   00           01   FB 0005B      CALLS   #1, LIB$SIGNAL                          ;
                                                     04   00062          RET                                             ; 3594
                                            08   BC  04   A2   D0 00063 5$:      MOVL    4(R2), @BOUNDS_DESC             ; 3606
                                                     04   00068          RET                                             ; 3608

; Routine Size:  105 bytes,    Routine Base:  _PAT$CODE + 06CC
```

```
; 1895          3609  1 END                           !End of module
; 1896          3610  0 ELUDOM


                                                .EXTRN  LIB$SIGNAL

;                               PSECT SUMMARY

;       Name                    Bytes                   Attributes

;   _PAT$OWN                      16  NOVEC,  WRT,  RD ,NOEXE,NOSHR,  LCL,  REL,  CON,NOPIC,ALIGN(2)
;   _PAT$CODE                   1845  NOVEC,NOWRT,  RD ,  EXE,NOSHR,  LCL,  REL,  CON,NOPIC,ALIGN(2)
;   _PAT$PLIT                    334  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       26         0      1000          00:01.7


; Information:  11
; Warnings:     0
; Errors:       0


;                                   COMMAND QUALIFIERS

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

; Size:         1845 code + 350 data bytes
; Run Time:       00:48.9
; Elapsed Time:   02:58.5
; Lines/CPU Min:   4430
; Lexemes/CPU-Min: 26326
; Memory Used:  284 pages
; Compilation Complete
```

PATARI
LIS

PATCMD
LIS

PATECO
LIS

PATCON
LIS

PATENC
LIS

PATBAS
LIS

PATBLD
LIS