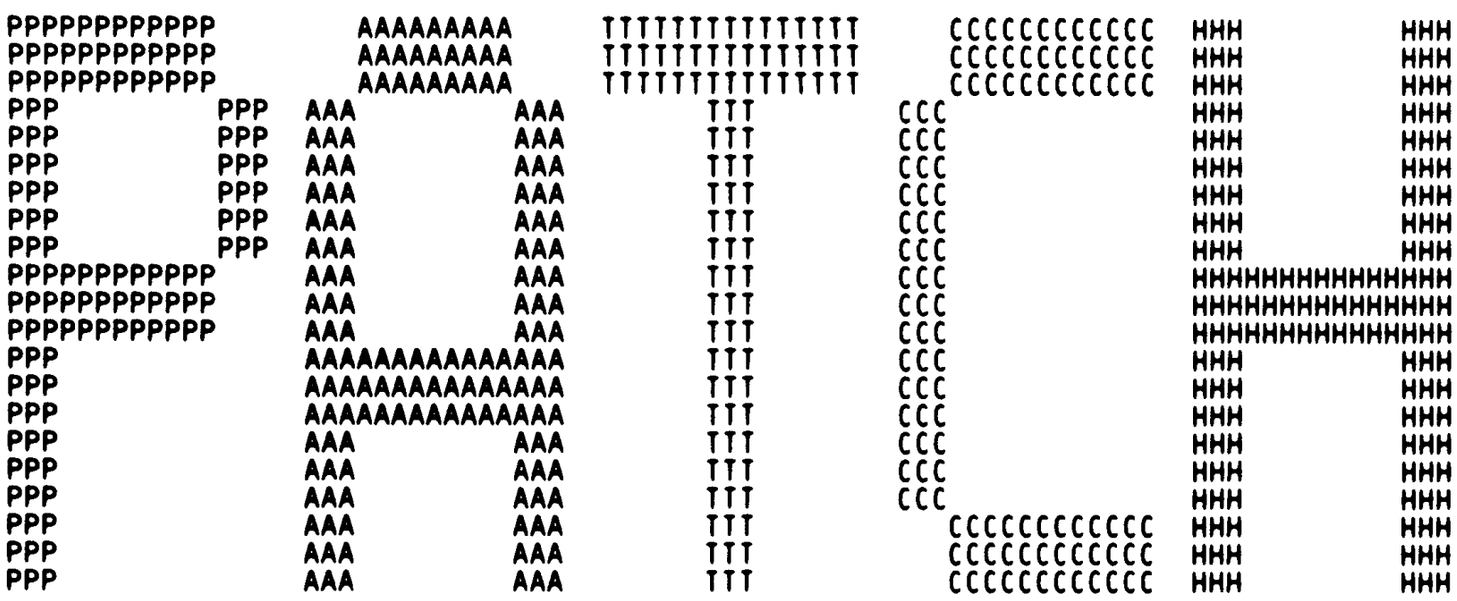
PPPPPPPPPPPP AAAAAAAAA TTTTTTTTTTTTTTT CCCCCCCCCCCC HHH HHH
PPP PPP AAA AAA TTT CCC HHH HHH
PPPPPPPPPPPP AAAAAAAAAAAAAAAA TTT CCC HHHHHHHHHHHHHHHH
PPP AAA AAA TTT CCC HHH HHH
PPP AAA AAA TTT CCCCCCCCCCCC HHH HHH

```
PPPPPPPP      AAAAAA    TTTTTTTTTT  RRRRRRRR    EEEEEEEEEE  BBBBBBBB
PPPPPPPP      AAAAAA    TTTTTTTTTT  RRRRRRRR    EEEEEEEEEE  BBBBBBBB
PP      PP   AA    AA       TT      RR      RR  EE          BB      BB
PP      PP   AA    AA       TT      RR      RR  EE          BB      BB
PP      PP   AA    AA       TT      RR      RR  EE          BB      BB
PP      PP   AA    AA       TT      RR      RR  EE          BB      BB
PPPPPPPP     AA    AA       TT      RRRRRRRR    EEEEEEEE    BBBBBBBB
PPPPPPPP     AA    AA       TT      RRRRRRRR    EEEEEEEE    BBBBBBBB
PP           AAAAAAAAAA     TT      RR  RR      EE          BB      BB
PP           AAAAAAAAAA     TT      RR  RR      EE          BB      BB
PP           AA    AA       TT      RR    RR    EE          BB      BB      ....
PP           AA    AA       TT      RR    RR    EE          BB      BB      ....
PP           AA    AA       TT      RR      RR  EEEEEEEEEE  BBBBBBBB        ....
PP           AA    AA       TT      RR      RR  EEEEEEEEEE  BBBBBBBB        ....

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 PATREB ( !
;  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:     RST routines used primarily to rebuild RST data structures.
; 38          0038  1 !
; 39          0039  1 !
; 40          0040  1 ! ENVIRONMENT:  This module runs on VAX under VAX/VMS, user mode, non-AST level.
; 41          0041  1 !
; 42          0042  1 ! Author:       Kevin Pammett, August 12, 1977.
; 43          0043  1 !
; 44          0044  1 ! Version:      V02-008
; 45          0045  1 !
; 46          0046  1 ! MODIFIED BY:
; 47          0047  1 !
; 48          0048  1 !       V03-001 MTR0012         Mike Rhodes             16-Aug-1982
; 49          0049  1 !               Modify file names to remove duplicate file name useage
; 50          0050  1 !               between code and require files.
; 51          0051  1 !
; 52          0052  1 !       V02-008 PCG0001         Peter George            02-FFB-1981
; 53          0053  1 !               Add require statement for LIB$:PATDEF.REQ
; 54          0054  1 !
; 55          0055  1 ! MODIFICATIONS:
; 56          0056  1 !
; 57          0057  1 !  NO   DATE            PROGRAMMER              PURPOSE
```

```
;    58          0058  !  --   ----          ----------          -------
;    59          0059  !
;    60          0060  !  00   22-DEC-77      K.D. MORSE          ADAPT VERSION 36 FOR PATCH.
;    61          0061  !  01   24-JAN-78      K.D. MORSE          ADD ERROR MSG PAT$_MODNOTADD (37).
;    62          0062  1  02   28-JAN-78      K.D. MORSE          SET_MODULE NOW WORKS FROM
;    63          0063  1  !                                       THE COMMAND ARGUMENT LIST
;    64          0064  1  !                                       OR FROM AN MC POINTER. (38)
;    65          0065  1  !                                       SET_MODULE AND CANC_MODULE
;    66          0066  1  !                                       UNDERSTAND /ALL. (39)
;    67          0067  1  !                                       THE SAT/LVT ACCESSING LOGIC WAS
;    68          0068  1  !                                       ADDED TO THE SORT FOR
;    69          0069  1  !                                       PERFORMANCE IMPROVEMENT. (40)
;    70          0070  1  !  03   04-APR-78      K.D. MORSE          NO CHANGES FOR 41.
;    71          0071  1  !  04   18-APR-78      K.D. MORSE          CHANGE NXT_SAT_LVT TO
;    72          0072  1  !                                       GET_NXT_SAT_LVT AND MADE ACCESS
;    73          0073  1  !                                       TO IT INCLUDE THE RECORD SIZE. 42
;    74          0074  1  !                                       PUT IN SET AND CANCEL /ALL MODU. 43
;    75          0075  1  !  05   25-APR-78      K.D. MORSE          CONVERT TO NATIVE COMPILER.
;    76          0076  1  !  06   18-MAY-78      K.D. MORSE          CANCEL MODULE ALSO NOT CAUSES
;    77          0077  1  !                                       SCOPE TO BE CANCELLED IF BOTH
;    78          0078  1  !                                       ARE THE SAME. (44)
;    79          0079  1  !                                       NO CHANGES FOR VERS 45.
;    80          0080  1  !  07   13-JUN-78      K.D. MORSE          ADD FAO COUNTS TO SIGNALS.
;    81          0081  1  !
;    82          0082  1  !--
```

```
;    84          0083  1 !
;    85          0084  1 ! TABLE OF CONTENTS:
;    86          0085  1 !
;    87          0086  1
;    88          0087  1 FORWARD ROUTINE
;    89          0088  1         PAT$GET_NXT_SAT,                        ! Provide access to the SAT
;    90          0089  1         PAT$GET_NXT_LVT,                        ! Provide access to the LVT
;    91          0090  1         GET_NXT_SAT_LVT,                            ! Parameterized access to SAT/LVT.
;    92          0091  1         PAT$SET_MODULE : NOVALUE,               ! Add a <module list> to the RST.
;    93          0092  1         PAT$CANC_MODULE : NOVALUE,              ! Delete a <module list> from the RST.
;    94          0093  1         PAT$SORT_SA_LVT : NOVALUE,              ! Sort the LVT or SAT vector.
;    95          0094  1         DELE_SAT_LVT,                           ! Mark SAT or LVT records for deletion.
;    96          0095  1         EMPTY_SAT_LVT : NOVALUE,                ! Remove the SAT/LVT portion of a
;    97          0096  1                                                 !  module from the RST.
;    98          0097  1         UNLINK_NT_RECS : NOVALUE;               ! Remove NTs from hash chains.
;    99          0098  1
;   100          0099  1 !
;   101          0100  1 ! INCLUDE FILES:
;   102          0101  1 !
;   103          0102  1
;   104          0103  1 LIBRARY 'SYS$LIBRARY:LIB.L32';
;   105          0104  1 REQUIRE 'SRC$:PATPCT.REQ';
;   106          0144  1 REQUIRE 'LIB$:PATDEF.REQ';                      ! Defines literals
;   107          0198  1 REQUIRE 'LIB$:PATMSG.REQ';
;   108          0372  1 REQUIRE 'SRC$:VXSMAC.REQ';
;   109          0437  1 REQUIRE 'SRC$:BSTRUC.REQ';
;   110          0513  1 REQUIRE 'SRC$:LISTEL.REQ';
;   111          0555  1 REQUIRE 'SRC$:PATGEN.REQ';
;   112          0777  1 REQUIRE 'SRC$:PATRTS.REQ';
;   113          1873  1 REQUIRE 'SRC$:SYSSER.REQ';
```

```
;  R1905  1      SWITCHES LIST (SOURCE);
;  R1906  1
;  R1907  1      EXTERNAL ROUTINE
;  R1908  1              PAT$fao_out;            ! formats a line and outputs to the terminal
;  R1909  1
```

```
;  114          1955  1
;  115          1956  1 !
;  116          1957  1 ! MACROS:
;  117          1958  1 !
;  118          1959  1
;  119          1960  1 !
;  120          1961  1 ! EQUATED SYMBOLS:
;  121          1962  1 !
;  122          1963  1
;  123          1964  1 !
;  124          1965  1 ! EXTERNAL REFERENCES:
;  125          1966  1 !
;  126          1967  1
;  127          1968  1 EXTERNAL
;  128          1969  1         PAT$GL_CONTEXT : BITVECTOR,                     ! CONTEXT QUALIFIER BITS (/ALL)
;  129          1970  1         PAT$GL_RST_BEGN,                                ! ADDRESS OF START OF RST
;  130          1971  1         PAT$GL_CSP_PTR : REF PATHNAME_VECTOR,           ! Pointer to the current SCOPE
;  131          1972  1         PAT$GL_MC_PTR  : REF MC_RECORD,                 ! Pointer to the Module Chain (MC).
;  132          1973  1         PAT$GL_HEAD_LST;                                ! pointer to expression arg list.
;  133          1974  1
;  134          1975  1
;  135          1976  1 EXTERNAL ROUTINE
;  136          1977  1         PAT$ADD_MODULE,                                 ! Add a module's symbols to the RST.
;  137          1978  1         PAT$VS_FREE : NOVALUE,                          ! Release vector storage.
;  138          1979  1         PAT$SAVE_SCOPE;                                 ! CANC MODU may imply CANC SCOPE
```

```
;  140          1980  1 GLOBAL ROUTINE PAT$GET_NXT_SAT( ACCESS_FLAG) =
;  141          1981  2 BEGIN
;  142          1982  2 RETURN( GET_NXT_SAT_LVT( .ACCESS_FLAG, RST_SAT_SIZE ));
;  143          1983  1 END;

                                                        .TITLE  PATREB
                                                        .IDENT  \V04-000\

                                                        .EXTRN  PAT$FAO_OUT, PAT$GL_CONTEXT
                                                        .EXTRN  PAT$GL_RST_BEGN
                                                        .EXTRN  PAT$GL_CSP_PTR, PAT$GL_MC_PTR
                                                        .EXTRN  PAT$GL_HEAD_LST
                                                        .EXTRN  PAT$ADD_MODULE, PAT$VS_FREE
                                                        .EXTRN  PAT$SAVE_SCOPE
                                                        .WEAK   ACCESS_CHECK

                                                        .PSECT  _PAT$CODE,NOWRT,2

                                  0000 00000            .ENTRY  PAT$GET_NXT_SAT, Save nothing                   ; 1980
                               0A   DD 00002            PUSHL   #10                                             ; 1982
                            04    AC   DD 00004         PUSHL   ACCESS_FLAG                                     ;
                00000000V   EF        02   FB 00007     CALLS   #2, GET_NXT_SAT_LVT                             ;
                                        04 0000E        RET                                                     ; 1983

; Routine Size:  15 bytes,    Routine Base:  _PAT$CODE + 0000

;  144          1984  1
;  145          1985  1
;  146          1986  1 GLOBAL ROUTINE PAT$GET_NXT_LVT( ACCESS_FLAG ) =
;  147          1987  2 BEGIN
;  148          1988  2 RETURN( GET_NXT_SAT_LVT( .ACCESS_FLAG, RST_LVT_SIZE ));
;  149          1989  1 END;


                                  0000 00000            .ENTRY  PAT$GET_NXT_LVT, Save nothing                   ; 1986
                               06   DD 00002            PUSHL   #6                                              ; 1988
                            04    AC   DD 00004         PUSHL   ACCESS_FLAG                                     ;
                00000000V   EF        02   FB 00007     CALLS   #2, GET_NXT_SAT_LVT                             ;
                                        04 0000E        RET                                                     ; 1989

; Routine Size:  15 bytes,    Routine Base:  _PAT$CODE + 000F
```

```
;  151          1990  1 ROUTINE GET_NXT_SAT_LVT( ACCESS_FLAG, RECORD_SIZE ) =
;  152          1991  1
;  153          1992  1 !++
;  154          1993  1 ! Functional Description:
;  155          1994  1 !
;  156          1995  1 !       Provide access to the RST SAT or LVT structures.
;  157          1996  1 !       i.e. Allow sequential access to the SAT/LVT via
;  158          1997  1 !       successive calls to this 'mapping' routine
;  159          1998  1 !       so that the accessor has no built-in notion of
;  160          1999  1 !       just how this sequential access is achieved.
;  161          2000  1 !
;  162          2001  1 ! Formal Parameters:
;  163          2002  1 !
;  164          2003  1 !       access_flag -   One of three (3) possible values which indicate
;  165          2004  1 !                       what type of access this is.
;  166          2005  1 !                       SA_ACCE_INIT    - Initialize the OWN pointers for further
;  167          2006  1 !                                         access.
;  168          2007  1 !                       SA_ACCE_RECS    - Return the next record in the structure.
;  169          2008  1 !                       SA_ACCE_FREE    - Return the next record marked for deletion
;  170          2009  1 !                                         that is not in a module about to be deleted.
;  171          2010  1 !       record_size -   The number of bytes in the record structure being accessed.
;  172          2011  1 !
;  173          2012  1 ! Implicit Inputs:
;  174          2013  1 !
;  175          2014  1 !       This routine defines how the SAT and LVT are accessed
;  176          2015  1 !       sequentially.  This definition must correspond to the one
;  177          2016  1 !       that is also 'built-in' in the routine DBG$SORT_SA_LVT.
;  178          2017  1 !       The latter does not use this routine's defintion because
;  179          2018  1 !       of the performance penality we pay to do it.  (all the CALLSes)
;  180          2019  1 !
;  181          2020  1 !       Some other routines also access the S'T/LVT 'on their own',
;  182          2021  1 !       but they restrict themselves to only looking at one storage vector's
;  183          2022  1 !       contribution to the SAT/LVT.
;  184          2023  1 !
;  185          2024  1 !       The MC storage descriptors for all MCs marked MC_IN_RST
;  186          2025  1 !       (this includes the one for globals) must be consistent.
;  187          2026  1 !
;  188          2027  1 ! Implicit Outputs:
;  189          2028  1 !
;  190          2029  1 !       none.
;  191          2030  1 !
;  192          2031  1 ! Routine Value:
;  193          2032  1 !
;  194          2033  1 !       0, when there are no more SAT or LVT records given the
;  195          2034  1 !          indicated ending criteria,
;  196          2035  1 !       Otherwise, a pointer to the appropriate record.
;  197          2036  1 !
;  198          2037  1 !       On the _INIT call, this routine returns the number of
;  199          2038  1 !       bytes in the SAT/LVT record which the _INIT
;  200          2039  1 !       was done for.
;  201          2040  1 !
;  202          2041  1 ! Side Effects:
;  203          2042  1 !
;  204          2043  1 !       The SAT or LVT is accessed sequentially.
;  205          2044  1 !       OWN storage is initialized and used.
;  206          2045  1 !--
;  207          2046  1
```

```
;  208     2047  2 BEGIN
;  209     2048  2
;  210     2049  2 LABEL
;  211     2050  2         NEXT_MC_LOOP;
;  212     2051  2
;  213     2052  2
;  214     2053  2 OWN
;  215     2054  2         LAST_PTR : REF SAT_RECORD,                      ! Pointer to the record we returned on
;  216     2055  2                                                        !  the previous call to this routine.
;  217     2056  2         MC_PTR : REF MC_RECORD;                         ! Module that LAST_PTR was in.
;  218     2057  2
;  219     2058  2 LOCAL
;  220     2059  2         STORE_DESC : REF VECT_STORE_DESC,               ! Pointer to the storage descriptor for the
;  221     2060  2                                                        !  type of storage we are currently looking
;  222     2061  2         CURRENT,                                        ! We keep two pointers to the SAT or LVT
;  223     2062  2                                                        ! record currently under consideration.
;  224     2063  2                                                        ! One, 'current', is NOT a SAT_POINTER becau
;  225     2064  2                                                        ! such longword pointers aren't easily compa
;  226     2065  2                                                        ! to the NI-pointers we get from storage des
;  227     2066  2         CURRENT_SAT_PTR : REF SAT_RECORD;               ! The other, 'current_sat_ptr' is a real SAT
;  228     2067  2                                                        ! pointer which always corresponds to 'curre
;  229     2068  2
;  230     2069  2 !++
;  231     2070  2 ! Initialization mode processing is simple and contained.  We simply set up a
;  232     2071  2 ! new context.
;  233     2072  2 !--
;  234     2073  3 IF (.access_flag EQL SL_ACCE_INIT)
;  235     2074  2 THEN
;  236     2075  3         BEGIN
;  237     2076  3         !++
;  238     2077  3         ! Set up so that the next call will find the
;  239     2078  3         ! first LVT or SAT record given the access criterion.
;  240     2079  3         !--
;  241     2080  3         MC_PTR = .PAT$GL_MC_PTR;
;  242     2081  3         LAST_PTR = 0;
;  243     2082  3         RETURN(.RECORD_SIZE);                           ! Return the record size so that we can
;  244     2083  3                                                         ! localize this knowledge to this routine.
;  245     2084  2         END;
;  246     2085  2
;  247     2086  2 !++
;  248     2087  2 ! Iterate to set 'current' and 'current_sat_ptr' to point to
;  249     2088  2 ! the next record we want to consider.
;  250     2089  2 !--
;  251     2090  2 REPEAT
;  252     2091  2     NEXT_MC_LOOP:
;  253     2092  3         BEGIN
;  254     2093  3         !++
;  255     2094  3         ! Once we run off the end of the module chain
;  256     2095  3         ! there is no more of any kind of storage.
;  257     2096  3         !--
;  258     2097  4         IF (.MC_PTR EQL 0) AND (.LAST_PTR EQL 0)
;  259     2098  3         THEN
;  260     2099  3                 RETURN(0);
;  261     2100  3
;  262     2101  3         !++
;  263     2102  3         ! See if this module is a possible candidate.
;  264     2103  3         !--
```

```
;  265          2104  3          IF .MC_PTR[MC_IN_RST]
;  266          2105  3          THEN
;  267          2106  4                  BEGIN
;  268          2107  4                  !++
;  269          2108  4                  ! The _FREE access type asks that we look only at
;  270          2109  4                  ! MCs which are both _IN_RST and NOT _DYING.
;  271          2110  5                  IF (.access_flag EQL SL_ACCE_FREE)
;  272          2111  4                  THEN
;  273          2112  4                          IF .MC_PTR[MC_IS_DYING]
;  274          2113  4                          THEN
;  275          2114  5                                  BEGIN
;  276          2115  5                                  !++
;  277          2116  5                                  ! This module is soon to be deleted
;  278          2117  5                                  ! so there is no _FREE storage therein.
;  279          2118  5                                  ! Skip over this module completely.
;  280          2119  5                                  !--
;  281          2120  5                                  MC_PTR = .MC_PTR [MC_NEXT];
;  282          2121  5                                  LEAVE NEXT_MC_LOOP;
;  283          2122  4                                  END;
;  284          2123  4
;  285          2124  4                  !++                                                                     ; R
;  286          2125  4                  ! See if this module has any of the kind
;  287          2126  4                  ! of storage we are looking for.
;  288          2127  4                  !--
;  289          2128  6                  STORE_DESC = (IF (.RECORD_SIZE EQL RST_SAT_SIZE)
;  290          2129  5                                THEN MC_PTR [MC_SAT_STORAGE]
;  291          2130  4                                ELSE MC_PTR[MC_VT_STORAGE]);
;  292          2131  4
;  293          2132  4                  !++
;  294          2133  4                  ! No storage is allocated if the 'begin' field is null.
;  295          2134  4                  !--
;  296          2135  5                  IF (.STORE_DESC [STOR_BEGIN_RST] NEQ 0)
;  297          2136  4                  THEN
;  298          2137  5                          BEGIN
;  299          2138  5                          !++
;  300          2139  5                          ! Found the right place to look.  Point
;  301          2140  5                          ! to the next record within this module.
;  302          2141  5                          !--
;  303          2142  6                          IF (.LAST_PTR EQL 0)
;  304          2143  5                          THEN
;  305          2144  5                                  !++
;  306          2145  5                                  ! First time thru we start at the beginning
;  307          2146  5                                  !--
;  308          2147  5                                  CURRENT = .STORE_DESC [STOR_BEGIN_RST]
;  309          2148  5                          ELSE
;  310          2149  5                                  !++
;  311          2150  5
;  312          2151  5                                  ! Subsequent times thru we assume that
;  313          2152  5                                  ! the records are contiguous and fixed-size.
;  314          2153  5                                  !--
;  315          2154  5                                  CURRENT = .LAST_PTR + .RECORD_SIZE;
;  316          2155  5
;  317          2156  5                          !++
;  318          2157  5                          ! The 'current' pointer must be within range and it
;  319          2158  5                          ! must point to a record we are interested in, given
;  320          2159  5                          ! the access criteria. Loop thru the storage for this
;  321          2160  5                          ! module as long as there are still some to look at.
```

```
;   322     2161  5                 !--
;   323     2162  6                 IF (
;   324     2163  8                 (WHILE( .CURRENT LSSA .STORE_DESC [STOR_MARKER] )
;   325     2164  7                         DO
;   326     2165  8                         BEGIN
;   327     2166  8                         !++
;   328     2167  8                         ! Initialize the SAT_POINTER which
;   329     2168  8                         ! corresponds to 'current'.
;   330     2169  8                         !--
;   331     2170  8                         CURRENT_SAT_PTR = .CURRENT + .PAT$GL_RST_BEGN;
;   332     2171  8
;   333     2172  8                         !++
;   334     2173  8                         ! As long as we are in range, we can imply skip
;   335     2174  8                         ! along successive records until we get one
;   336     2175  8                         ! which satisfies the given access criteria.
;   337     2176  8                         !--
;   338     2177  9                         IF (.access_flag EQL SL_ACCE_RECS)
;   339     2178  8                         THEN
;   340     2179  9                                 BEGIN
;   341     2180  9                                 !++
;   342     2181  9                                 ! Skip records which are marked
;   343     2182  9                                 ! for deletion.  Anything else is OK.
;   344     2183  9                                 !--
;   345     2184 10                                 IF (.CURRENT_SAT_PTR [SAT_NT_PTR] NEQ 1)
;   346     2185  9                                 THEN
;   347     2186  9                                         EXITLOOP(TRUE);
;   348     2187  9                                 END
;   349     2188  8                         ELSE
;   350     2189  8
;   351     2190  9                                 BEGIN
;   352     2191  9                                 !++
;   353     2192  9                                 ! The _FREE mode asks only for deleted
;   354     2193  9                                 ! records in NOT _DYING modules.
;   355     2194  9                                 !--
;   356     2195 10                                 IF (.CURRENT_SAT_PTR [SAT_NT_PTR] EQL 1)
;   357     2196  9                                 THEN
;   358     2197  9                                         EXITLOOP(TRUE);
;   359     2198  8                                 END;
;   360     2199  8
;   361     2200  8                         !++
;   362     2201  8                         ! Skip along to the next record.
;   363     2202  8                         !--
;   364     2203  8                         CURRENT = .CURRENT + .RECORD_SIZE;
;   365     2204  8                         END
;   366     2205  6                 ) EQL TRUE
;   367     2206  6                    )
;   368     2207  5                 THEN
;   369     2208  5                         !++
;   370     2209  5                         ! We have found a valid record in the current module.
;   371     2210  5                         !--
;   372     2211  5                         EXITLOOP;
;   373     2212  5
;   374     2213  5                 !++
;   375     2214  5                 ! If we fall out of the above loop, there were no records
;   376     2215  5                 ! in the current module which satsified the access
;   377     2216  5                 ! criteria. Set up so that when we look at the next
;   378     2217  5                 ! module we will begin at the first record.
```

```
;  379          2218  5                        !--
;  380          2219  5                        LAST_PTR = 0;
;  381          2220  4                        END;
;  382          2221  4
;  383          2222  4                    !++
;  384          2223  4                    ! There is none of the kind of storage we are looking for so
;  385          2224  4                    ! loop back to consider the next module.
;  386          2225  4                    !--
;  387          2226  3                    END;
;  388          2227  3
;  389          2228  3            !++
;  390          2229  3            ! The current module is not in the RST so loop back to consider the next one.
;  391          2230  3            !--
;  392          2231  3            MC_PTR = .MC_PTR [MC_NEXT];
;  393          2232  2            END;
;  394          2233  2
;  395          2234  2 !++
;  396          2235  2 ! Pass back the desired pointer having first stored it away in our OWN storage
;  397          2236  2 ! so that we can pick it up on the next call.
;  398          2237  2 !--
;  399          2238  2 LAST_PTR = .CURRENT;
;  400          2239  2 RETURN(.CURRENT_SAT_PTR);
;  401          2240  1 END;


                                                .PSECT  _PAT$OWN,NOEXE,2

                                        00000 LAST_PTR:
                                                .BLKB   4
                                        00004 MC_PTR:   .BLKB   4


                                                .PSECT  _PAT$CODE,NOWRT,2

                                        007C 00000 GET_NXT_SAT_LVT:
                                                .WORD   Save R2,R3,R4,R5,R6                     ; 1990
                    56 00000000'    EF  9E 00002        MOVAB   MC_PTR, R6
                    54      04      AC  D0 00009        MOVL    ACCESS_FLAG, R4                 ; 2073
                                    0F  12 0000D        BNEQ    1$
                    66 00000000G    EF  D0 0000F        MOVL    PAT$GL_MC_PTR, MC_PTR           ; 2080
                            FC      A6  D4 00016        CLRL    LAST_PTR                        ; 2081
                    50      08      AC  D0 00019        MOVL    RECORD_SIZE, R0                 ; 2082
                                        04 0001D        RET
                    51              66  D0 0001E 1$:     MOVL    MC_PTR, R1                      ; 2097
                                    08  12 00021        BNEQ    2$
                            FC      A6  D5 00023        TSTL    LAST_PTR
                                    03  12 00026        BNEQ    2$
                                008C    31 00028        BRW     16$
                    52 00000000G    EF  D0 0002B 2$:     MOVL    PAT$GL_RST_BEGN, R2             ; 2104
                    51              52  C0 00032        ADDL2   R2, R1
6B          03      A1              01  E1 00035        BBC     #1, 3(R1), 13$
                    03              54  D1 0003A        CMPL    R4, #3                          ; 2110
                                    05  12 0003D        BNEQ    3$
65          03      A1              06  E0 0003F        BBS     #6, 3(R1), 14$                  ; 2112
                    0A      08      AC  D1 00044 3$:     CMPL    RECORD_SIZE, #10                ; 2128
```

```
                                        0A  12 00048          BNEQ    4$                                         :
                   51         66        52  C1 0004A          ADDL3   R2, MC_PTR, R1                             : 2129
                              50    23  A1  9E 0004E          MOVAB   35(R1), STORE_DESC                         :
                                        08  11 00052          BRB     5$                                         :
                   51         66        52  C1 00054 4$:      ADDL3   R2, MC_PTR, R1                             : 2130
                              50    2A  A1  9E 00058          MOVAB   42(R1), STORE_DESC                         :
                                    01  A0  B5 0005C 5$:      TSTW    1(STORE_DESC)                              : 2135
                                        44  13 0005F          BEQL    13$                                        :
                              51    FC  A6  D0 00061          MOVL    LAST_PTR, R1                               : 2142
                                        06  12 00065          BNEQ    6$                                         :
                              53    01  A0  3C 00067          MOVZWL  1(STORE_DESC), CURRENT                     : 2147
                                        05  11 0006B          BRB     7$                                         :
                   53         51    08  AC  C1 0006D 6$:      ADDL3   RECORD_SIZE, R1, CURRENT                   : 2154
     53      05    A0         10        00  ED 00072 7$:      CMPZV   #0, #16, 5(STORE_DESC), CURRENT            : 2163
                                        20  1B 00078          BLEQU   11$                                        :
                   55         53        52  C1 0007A          ADDL3   R2, CURRENT, CURRENT_SAT_PTR               : 2170
                              01        54  D1 0007E          CMPL    R4, #1                                     : 2177
                                        07  12 00081          BNEQ    8$                                         :
                              01        65  B1 00083          CMPW    (CURRENT_SAT_PTR), #1                      : 2184
                                        0C  13 00086          BEQL    10$                                        :
                                        05  11 00088          BRB     9$                                         : 2186
                              01        65  B1 0008A 8$:      CMPW    (CURRENT_SAT_PTR), #1                      : 2195
                                        05  12 0008D          BNEQ    10$                                        :
                              51        01  D0 0008F 9$:      MOVL    #1, R1                                     : 2197
                                        09  11 00092          BRB     12$                                        :
                              53    08  AC  C0 00094 10$:     ADDL2   RECORD_SIZE, CURRENT                       : 2203
                                        D8  11 00098          BRB     7$                                         : 2163
                              51        01  CE 0009A 11$:     MNEGL   #1, R1                                     :
                              01        51  D1 0009D 12$:     CMPL    R1, #1                                     : 2205
                                        0D  13 000A0          BEQL    15$                                        :
                                    FC  A6  D4 000A2          CLRL    LAST_PTR                                   : 2219
                   51         66        52  C1 000A5 13$:     ADDL3   R2, MC_PTR, R1                             : 2231
                              66        61  3C 000A9 14$:     MOVZWL  (R1), MC_PTR                               :
                                      FF6F  31 000AC          BRW     1$                                         : 2084
                        FC    A6        53  D0 000AF 15$:     MOVL    CURRENT, LAST_PTR                          : 2238
                              50        55  D0 000B3          MOVL    CURRENT_SAT_PTR, R0                        : 2239
                                            04 000B6          RET                                                :
                                        50  D4 000B7 16$:     CLRL    R0                                         : 2240
                                            04 000B9          RET                                                :

; Routine Size:  186 bytes,    Routine Base:  _PAT$CODE + 001E
```

```
;  403     2241  1 GLOBAL ROUTINE PAT$SET_MODULE (MODULE_TO_SET) : NOVALUE =
;  404     2242  1
;  405     2243  1 !++
;  406     2244  1 ! Functional Description:
;  407     2245  1 !
;  408     2246  1 !       Add the symbols for a given list of modules, or /ALL modules to the RST.
;  409     2247  1 !
;  410     2248  1 ! Formal Parameters:
;  411     2249  1 !
;  412     2250  1 !       MODULE_TO_SET   -an MC pointer to the module we are
;  413     2251  1 !                        to add, or 0 => use the command
;  414     2252  1 !                        argument list or do SET /ALL.
;  415     2253  1 !
;  416     2254  1 !
;  417     2255  1 ! Implicit Inputs:
;  418     2256  1 !
;  419     2257  1 !       The expression list contains MC_PTRs to the indicated modules.
;  420     2258  1 !       If "SET MODULE /ALL" was specified, then the Module Chain points to
;  421     2259  1 !       all the modules to set.
;  422     2260  1 !
;  423     2261  1 ! Implicit Outputs:
;  424     2262  1 !
;  425     2263  1 !       The modules are added.
;  426     2264  1 !
;  427     2265  1 ! Return Value:
;  428     2266  1 !
;  429     2267  1 !       NOVALUE - This routine SIGNALs out if an error occurs.
;  430     2268  1 !
;  431     2269  1 ! Side Effects:
;  432     2270  1 !
;  433     2271  1 !       The symbols for the indicated modules get added to the RST.  If they
;  434     2272  1 !        are already there,  PATCH merely returns.
;  435     2273  1 !       If an error occurs, a SIGNAL and UNWIND are performed.
;  436     2274  1 !--
;  437     2275  1
;  438     2276  2 BEGIN
;  439     2277  2
;  440     2278  2 LOCAL
;  441     2279  2         DO_SAT_SORT,                                    ! Flags to control sorting.
;  442     2280  2         DO_LVT_SORT,
;  443     2281  2         POINTER,                                        ! Used to scan the arg list.
;  444     2282  2         MC_CHAIN : REF MC_RECORD,                       ! POINTER TO CURRENT MODULE CHAIN ENTRY
;  445     2283  2         MC_PTR : REF MC_RECORD,                         ! Each arg is an MC pointer.
;  446     2284  2         STORE_DESC : REF VECT_STORE_DESC;               ! We look at the storage descriptor
;  447     2285  2                                                         ! associated with a module after we
;  448     2286  2                                                         ! try to add it to see if SAT or LVT
;  449     2287  2                                                         ! storage was allocated.
;  450     2288  2
;  451     2289  2 !++
;  452     2290  2 ! Assume that we won't have to (re)sort the LVT and SAT - an added module doesn't
;  453     2291  2 ! necessarily add anything to the LVT or SAT.
;  454     2292  2 !--
;  455     2293  2 DO_SAT_SORT = FALSE;
;  456     2294  2 DO_LVT_SORT = FALSE;
;  457     2295  2 MC_CHAIN = 0;
;  458     2296  2
;  459     2297  2 !++
```

```
;  460          2298  2 ! If we were given a specific module_to_set, then we just assume that that is
;  461          2299  2 ! the MC_PTR we want.  Otherwise either DBG$GL_HEAD_LST points to an arg
;  462          2300  2 ! list of MC_PTRs, or it is 0, which means that we are to try to set ALL modules.
;  463          2301  2 !--
;  464          2302  3 IF (.MODULE_TO_SET EQL 0)
;  465          2303  2 THEN
;  466          2304  3         IF ((POINTER = .PAT$GL_HEAD_LST) EQL 0)
;  467          2305  2         THEN
;  468          2306  2                 MC_PTR = MC_CHAIN = .PAT$GL_MC_PTR;
;  469          2307  2
;  470          2308  2 !++
;  471          2309  2 ! Loop thru the arg list or module chain trying to add the indicated modules to
;  472          2310  2 ! the RST.  Note that we skip over the MC reserved for globals if the MC chain
;  473          2311  2 ! is being used.
;  474          2312  2
;  475          2313  2 !--
;  476          2314  2 REPEAT
;  477          2315  3         BEGIN
;  478          2316  3         !++
;  479          2317  3         ! PICK UP A POINTER TO THE MODULE TO BE SET.
;  480          2318  3         !--
;  481          2319  4         IF (.MC_CHAIN NEQ 0)
;  482          2320  3         THEN
;  483          2321  4                 BEGIN
;  484          2322  4                 !++
;  485          2323  4                 ! THE NEXT MODULE COMES FROM THE MODULE CHAIN.  CHECK FOR END
;  486          2324  4                 ! OF THE CHAIN.  This also skips the MC for globals.
;  487          2325  4                 !--
;  488          2326  5                 IF ((MC_PTR = .MC_PTR[MC_NEXT]) EQL 0)
;  489          2327  4                 THEN
;  490          2328  4                         EXITLOOP;
;  491          2329  4                 END
;  492          2330  3         ELSE
;  493          2331  3                 !++
;  494          2332  3                 ! THE NEXT MODULE IS EITHER ONE SPECIFICALLY REQUESTED OR
;  495          2333  3                 ! COMES FROM THE COMMAND ARGUMENT LIST.
;  496          2334  3                 !--
;  497          2335  4                 IF ((MC_PTR = .MODULE_TO_SET) EQL 0)
;  498          2336  3                 THEN
;  499          2337  3                         !++
;  500          2338  3                         ! EACH ARGUMENT IS A POINTER TO THE MC RECORD FOR
;  501          2339  3                         ! THE MODULE TO BE SET.
;  502          2340  3                         !--
;  503          2341  3                         MC_PTR = .LIST_ELEM_EXP1(.POINTER);
;  504          2342  3
;  505          2343  3         !++
;  506          2344  3         ! NOW CHECK THAT THE MODULE IS NOT ALREADY IN THE RST.
;  507          2345  3         !--
;  508          2346  4         IF ( NOT .MC_PTR[MC_IN_RST])
;  509          2347  3         THEN
;  510          2348  4                 BEGIN
;  511          2349  4                 !++
;  512          2350  4                 ! Simply add the module the same way that the RST init
;  513          2351  4                 ! procedure would.
;  514          2352  4                 !--
;  515          2353  5                 IF (PAT$ADD_MODULE( .MC_PTR ))
;  516          2354  4                 THEN
```

```
;  517     2355  5                    BEGIN
;  518     2356  5                    !++
;  519     2357  5                    ! Note that the module has been added, and see
;  520     2358  5                    ! if any LVT and/or SAT storage was allocated
;  521     2359  5                    ! for it to avoid an unnecessary SORT later on.
;  522     2360  5                    !--
;  523     2361  5                    MC_PTR [MC_IN_RST] = TRUE;
;  524     2362  5                    MC_PTR [MC_IS_DYING] = FALSE;
;  525     2363  5                    STORE_DESC = MC_PTR [MC_SAT_STORAGE];
;  526     2364  6                    IF (.STORE_DESC [STOR_BEGIN_RST] NEQ 0)
;  527     2365  5                    THEN
;  528     2366  5                            DO_SAT_SORT = TRUE;
;  529     2367  5                    STORE_DESC = MC_PTR [MC_LVT_STORAGE];
;  530     2368  6                    IF (.STORE_DESC [STOR_BEGIN_RST] NEQ 0)
;  531     2369  5                    THEN
;  532     2370  5                            DO_LVT_SORT = TRUE;
;  533     2371  5                    END
;  534     2372  4            ELSE
;  535     2373  4                    !++
;  536     2374  4                    ! IF A LIST OF MODULES IS BEING SET, THIS MESSAGE MUST
;  537     2375  4                    ! BE INFORMATIONAL BECAUSE A "RETURN" MUST BE EXECUTED
;  538     2376  4                    ! FROM HERE TO DO THE POST-MODULE "SET" CLEANUP.
;  539     2377  4                    ! OTHERWISE A SIGNAL CAN BE PERFORMED HERE AND OUT OF
;  540     2378  4                    ! THE "SET SCOPE" COMMAND AS WELL.  BY SUBTRACTING
;  541     2379  4                    ! ONE FROM THE INFORMATION ERROR CODE, PATCH CREATES
;  542     2380  4                    ! A WARNING CODE.
;  543     2381  4                    !--
;  544     2382  5                    IF (.MODULE_TO_SET NEQ 0)
;  545     2383  4                    THEN
;  546     2384  4                            SIGNAL(PAT$_MODNOTADD-1, 1, MC_PTR[MC_NAME_CS])
;  547     2385  4                    ELSE
;  548     2386  4                            SIGNAL(PAT$_MODNOTADD, 1, MC_PTR[MC_NAME_CS]);
;  549     2387  3                    END;
;  550     2388  3
;  551     2389  3                    !++
;  552     2390  3                    ! Go back and look at the next arg if there is one.
;  553     2391  3                    !--
;  554     2392  4                    IF (.MODULE_TO_SET NEQ 0)
;  555     2393  3                    THEN
;  556     2394  3                            !++
;  557     2395  3                            ! THERE IS NEVER A NEXT ONE IF ONLY ONE WAS SPECIFIED.
;  558     2396  3                            !--
;  559     2397  3                            EXITLOOP;
;  560     2398  3
;  561     2399  3                    !++
;  562     2400  3                    ! A NEXT ONE MUST COME FROM THE MC CHAIN OR THE ARG LIST.
;  563     2401  3                    !--
;  564     2402  4                    IF (.MC_CHAIN EQL 0)
;  565     2403  3                    THEN
;  566     2404  4                            IF ((POINTER = .LIST_ELEM_FLINK(.POINTER)) EQL 0)
;  567     2405  3                            THEN
;  568     2406  3                                    EXITLOOP;
;  569     2407  2                    END;
;  570     2408  2
;  571     2409  2 !++
;  572     2410  2 ! If any SAT/LVT storage was allocated, then corresponding records have been
;  573     2411  2 ! added so we have to re-sort the structure.
```

```
;  574          2412  2 !--
;  575          2413  3 IF (.DO_SAT_SORT)
;  576          2414  2 THEN
;  577          2415  2         PAT$SORT_SA_LVT( PAT$GET_NXT_SAT );
;  578          2416  3 IF (.DO_LVT_SORT)
;  579          2417  2 THEN
;  580          2418  2         PAT$SORT_SA_LVT( PAT$GET_NXT_LVT );
;  581          2419  2
;  582          2420  1 END;
```

```
                                07FC 00000          .ENTRY  PAT$SET_MODULE, Save R2,R3,R4,R5,R6,R7,R8,- ; 2241
                                                            R9,R10
                    5A 00000000V   EF   9E 00002    MOVAB   PAT$SORT_SA_LVT, R10                        ;
                    59 00000000G   EF   9E 00009    MOVAB   PAT$GL_RST_BEGN, R9                         ;
                                   57   7C 00010    CLRQ    DO_LVT_SORT                                 ; 2294
                                   56   D4 00012    CLRL    MC_CHAIN                                    ; 2295
                    55         04  AC   D0 00014    MOVL    MODULE_TO_SET, R5                           ; 2302
                                   13   12 00018    BNEQ    1$                                          ;
                    54 00000000G   EF   D0 0001A    MOVL    PAT$GL_HEAD_LST, POINTER                    ; 2304
                                   0A   12 00021    BNEQ    1$                                          ;
                    56 00000000G   EF   D0 00023    MOVL    PAT$GL_MC_PTR, MC_CHAIN                     ; 2306
                    52             56   D0 0002A    MOVL    MC_CHAIN, MC_PTR                            ;
                                   56   D5 0002D 1$: TSTL    MC_CHAIN                                    ; 2319
                                   0C   13 0002F    BEQL    2$                                          ;
                 50 52             69   C1 00031    ADDL3   PAT$GL_RST_BEGN, MC_PTR, R0                 ; 2326
                    52             60   3C 00035    MOVZWL  (R0), MC_PTR                                ;
                                   0C   12 00038    BNEQ    3$                                          ;
                                 0080   31 0003A    BRW     10$                                         ; 2328
                    52             55   D0 0003D 2$: MOVL    R5, MC_PTR                                  ; 2335
                                   04   12 00040    BNEQ    3$                                          ;
                    52         04  A4   D0 00042    MOVL    4(POINTER), MC_PTR                          ; 2341
                 50 52             69   C1 00046 3$: ADDL3   PAT$GL_RST_BEGN, MC_PTR, R0                 ; 2346
                 5E      03     A0   01   E0 0004A    BBS     #1, 3(R0), 8$                               ;
                                   52   DD 0004F    PUSHL   MC_PTR                                      ; 2353
                    00000000G   EF   01   FB 00051    CALLS   #1, PAT$ADD_MODULE                          ;
                                   27   50   E9 00058    BLBC    R0, 5$                                  ;
                 50 52             69   C1 0005B    ADDL3   PAT$GL_RST_BEGN, MC_PTR, R0                 ; 2361
                          03     A0   02   88 0005F    BISB2   #2, 3(R0)                                   ;
                          03     A0   40   8F 8A 00063 BICB2   #64, 3(R0)                                 ; 2362
                    53         23  A0   9E 00068    MOVAB   35(R0), STORE_DESC                          ; 2363
                               01  A3   B5 0006C    TSTW    1(STORE_DESC)                               ; 2364
                                   03   13 0006F    BEQL    4$                                          ;
                    58             01   D0 00071    MOVL    #1, DO_SAT_SORT                             ; 2366
                    53         2A  A0   9E 00074 4$: MOVAB   42(R0), STORE_DESC                          ; 2367
                               A3  B5 00078    TSTW    1(STORE_DESC)                               ; 2368
                                   30   13 0007B    BEQL    8$                                          ;
                    57             01   D0 0007D    MOVL    #1, DO_LVT_SORT                             ; 2370
                                   2B   11 00080    BRB     8$                                          ; 2353
                                   55   D5 00082 5$: TSTL    R5                                          ; 2382
                                   11   13 00084    BEQL    6$                                          ;
                 50 52             69   C1 00086    ADDL3   PAT$GL_RST_BEGN, MC_PTR, R0                 ; 2384
                               0C  A0   9F 0008A    PUSHAB  12(R0)                                      ;
                                   01   DD 0008D    PUSHL   #1                                          ;
```

```
                    006D800A   8F   DD 0008F           PUSHL   #7176202                                    ;
                               0F   11 00095           BRB     7$                                          ;
                50             52   69   C1 00097 6$:  ADDL3   PAT$GL_RST_BEGN, MC_PTR, R0                ; 2386
                                 0C A0   9F 0009B           PUSHAB  12(R0)                                  ;
                               01   DD 0009E           PUSHL   #1                                          ;
                    006D800B   8F   DD 000A0           PUSHL   #7176203                                    ;
            00000000G  00          03   FB 000A6 7$:  CALLS   #3, LIB$SIGNAL                              ;
                               55   D5 000AD 8$:  TSTL    R5                                              ; 2392
                               0C   12 000AF           BNEQ    10$                                         ;
                               56   D5 000B1           TSTL    MC_CHAIN                                    ; 2402
                               03   12 000B3           BNEQ    9$                                          ;
                54             64   D0 000B5           MOVL    (POINTER), POINTER                          ; 2404
                               03   13 000B8 9$:  BEQL    10$                                             ;
                             FF70   31 000BA           BRW     1$                                          ;
                07             58   E9 000BD 10$:  BLBC    DO_SAT_SORT, 11$                            ; 2413
                         FE64  CF   9F 000C0           PUSHAB  PAT$GET_NXT_SAT                             ; 2415      ; R
                6A             01   FB 000C4           CALLS   #1, PAT$SORT_SA_LVT                         ;
                07             57   E9 000C7 11$:  BLBC    DO_LVT_SORT, 12$                            ; 2416
                         FE69  CF   9F 000CA           PUSHAB  PAT$GET_NXT_LVT                             ; 2418
                6A             01   FB 000CE           CALLS   #1, PAT$SORT_SA_LVT                         ;
                               04   000D1 12$:  RET                                                      ; 2420

; Routine Size:  210 bytes,    Routine Base:  _PAT$CODE + 00D8
```

```
;  584     2421  1 GLOBAL ROUTINE PAT$CANC_MODULE : NOVALUE =
;  585     2422  1
;  586     2423  1 !++
;  587     2424  1 ! Functional Description:
;  588     2425  1 !
;  589     2426  1 !       Delete the symbols for a given list of module names from the RST,
;  590     2427  1 !       or delete /ALL modules.  If the SCOPE is set to a module that is to be
;  591     2428  1 !       cancelled, then the SCOPE is also cancelled.
;  592     2429  1 !
;  593     2430  1 ! Implicit Inputs:
;  594     2431  1 !
;  595     2432  1 !       Context bit if /ALL was specified or not.
;  596     2433  1 !       The expression list contains MC_PTRs to the indicated modules.
;  597     2434  1 !
;  598     2435  1 ! Implicit Outputs:
;  599     2436  1 !
;  600     2437  1 !       none.
;  601     2438  1 !
;  602     2439  1 ! Return Value:
;  603     2440  1 !
;  604     2441  1 !       NOVALUE
;  605     2442  1 !
;  606     2443  1 ! Side Effects:
;  607     2444  1 !
;  608     2445  1 !       The symbols for the indicated modules are deleted from the RST.
;  609     2446  1 !       If the symbols are not set, PATCH merely returns.  If an error occurs,
;  610     2447  1 !       then a SIGNAL and UNWIND are performed.
;  611     2448  1 !--
;  612     2449  1
;  613     2450  2 BEGIN
;  614     2451  2
;  615     2452  2 LOCAL
;  616     2453  2         MC_CHAIN : REF MC_RECORD,                       ! Pointer to scan module chain
;  617     2454  2         POINTER,                                        ! Used to scan the arg list.
;  618     2455  2         SHRINK_LVT,                                     ! Number of LVT records freed up.
;  619     2456  2         SHRINK_SAT,                                     ! Number of SAT records freed up.
;  620     2457  2         MC_PTR : REF MC_RECORD;
;  621     2458  2
;  622     2459  2 !++
;  623     2460  2 ! If the module names come from the command argument list, then check the
;  624     2461  2 ! consistency of the list.  Otherwise, initialize in preparation for the
;  625     2462  2 ! "CANCEL MODULE /ALL" command.
;  626     2463  2 !--
;  627     2464  2 MC_CHAIN = 0;
;  628     2465  3 IF ((POINTER = .PAT$GL_HEAD_LST) EQL 0)
;  629     2466  2 THEN
;  630     2467  2         MC_PTR = MC_CHAIN = .PAT$GL_MC_PTR;
;  631     2468  2
;  632     2469  2 !++
;  633     2470  2 ! To avoid having to shrink the SAT or LVT when unnecessary, initialize
;  634     2471  2 ! the flags (actually, counts) which indicate whether or not any records have
;  635     2472  2 ! been removed from the corresponding data structures.
;  636     2473  2 !--
;  637     2474  2 SHRINK_LVT = 0;
;  638     2475  2 SHRINK_SAT = 0;
;  639     2476  2
;  640     2477  2 !++
```

```
;  641          2478  2 ! Loop thru the arg list trying to delete the indicated modules from the RST.
;  642          2479  2 !--
;  643          2480  2 REPEAT
;  644          2481  3         BEGIN
;  645          2482  4         IF (.MC_CHAIN NEQ 0)
;  646          2483  3         THEN
;  647          2484  4                 BEGIN
;  648          2485  4                 !++
;  649          2486  4                 ! The next module name comes from the module chain.  Check
;  650          2487  4                 ! for the end of the chain.
;  651          2488  4                 !--
;  652          2489  5                 IF ((MC_PTR = .MC_PTR[MC_NEXT]) EQL 0)
;  653          2490  4                 THEN
;  654          2491  4                         EXITLOOP;
;  655          2492  4                 END
;  656          2493  3         ELSE
;  657          2494  3                 !++
;  658          2495  3                 ! The next module name comes from the command argument list.
;  659          2496  3                 ! Each argument is a pointer to the MC record for the module
;  660          2497  3                 ! to be deleted.
;  661          2498  3                 !--
;  662          2499  3                 MC_PTR = .LIST_ELEM_EXP1(.POINTER);
;  663          2500  3
;  664          2501  3         !++
;  665          2502  3         ! Don't try to delete one which is not already there,
;  666          2503  3         ! and don't complain if we are asked to.
;  667          2504  3         !--
;  668          2505  4         IF (.MC_PTR [MC_IN_RST])
;  669          2506  3         THEN
;  670          2507  4                 BEGIN
;  671          2508  4                 !++
;  672          2509  4                 ! Zero out all LVT or SAT records which correspond
;  673          2510  4                 ! to NT records which we are about to release.
;  674          2511  4                 !--
;  675          2512  4                 SHRINK_SAT = .SHRINK_SAT +
;  676          2513  4                         DELE_SAT_LVT( MC_PTR [MC_NT_STORAGE], PAT$GET_NXT_SAT );
;  677          2514  4                 SHRINK_LVT = .SHRINK_LVT +
;  678          2515  4                         DELE_SAT_LVT( MC_PTR [MC_NT_STORAGE], PAT$GET_NXT_LVT );
;  679          2516  4
;  680          2517  4
;  681          2518  4                 !++
;  682          2519  4                 ! If we have just cancelled a module into which the current
;  683          2520  4                 ! SCOPE is pointing, we cancel that scope so as th let the
;  684          2521  4                 ! user know it is useless.
;  685          2522  4                 !--
;  686          2523  5                 IF (.PAT$GL_CSP_PTR NEQA 0)
;  687          2524  4                 THEN
;  688          2525  5                         BEGIN
;  689          2526  5                         LOCAL
;  690          2527  5                                 CS_PTR : CS_POINTER;
;  691          2528  5
;  692          2529  5                         !++
;  693          2530  5                         ! Pick up the first entry in the CSP pathname vector
;  694          2531  5                         ! since that is the module name and is all we have
;  695          2532  5                         ! to look at to see if the CSP "points into" the
;  696          2533  5                         ! module we have just cancelled.
;  697          2534  5                         !--
```

```
;  698     2535  5                    CS_PTR = .PAT$GL_CSP_PTR[0];
;  699     2536  6                    IF (CH$EQL(.CS_PTR[0], CS_PTR[1], .MC_PTR[MC_NAME_CS],
;  700     2537  6                            MC_PTR[MC_NAME_ADDR]))
;  701     2538  5                    THEN
;  702     2539  5                            PAT$SAVE_SCOPE(FALSE);          ! Cancel the SCOPE
;  703     2540  4                    END;
;  704     2541  4                !++
;  705     2542  4                ! Note that the module is about to be cancelled.  We can't turn
;  706     2543  4                ! off MC_IN_RST yet because the access functions still use that
;  707     2544  4                ! flag and there are still valid records in the storage
;  708     2545  4                ! associated with this module.
;  709     2546  4                !--
;  710     2547  4                MC_PTR [MC_IS_DYING] = TRUE;
;  711     2548  4
;  712     2549  4                !++
;  713     2550  4                ! Unlink all NT records for this module from the hash chains
;  714     2551  4                ! they are currently in, and release the vector storage taken
;  715     2552  4                ! by them.
;  716     2553  4                !--
;  717     2554  4                UNLINK_NT_RECS( .MC_PTR );
;  718     2555  4                PAT$VS_FREE( MC_PTR [MC_NT_STORAGE] );
;  719     2556  3                END;
;  720     2557  3        !++
;  721     2558  3        ! Check if there is another module name.
;  722     2559  3        !--
;  723     2560  4        IF (.MC_CHAIN EQL 0)
;  724     2561  3        THEN
;  725     2562  4                IF ((POINTER = .LIST_ELEM_FLINK(.POINTER)) EQL 0)
;  726     2563  3                THEN
;  727     2564  3                        EXITLOOP;
;  728     2565  2                END;
;  729     2566  2
;  730     2567  2        !++
;  731     2568  2        ! For each module to be cancelled, we still have associated SAT/LVT storage
;  732     2569  2        ! which probably contains records that correspond to other modules.  We must
;  733     2570  2        ! move these to safer ground and then actually free up the storage.
;  734     2571  2        !--
;  735     2572  3        IF (.SHRINK_SAT NEQ 0)
;  736     2573  2        THEN
;  737     2574  2                EMPTY_SAT_LVT( PAT$GET_NXT_SAT );
;  738     2575  3        IF (.SHRINK_LVT NEQ 0)
;  739     2576  2        THEN
;  740     2577  2                EMPTY_SAT_LVT( PAT$GET_NXT_LVT );
;  741     2578  2
;  742     2579  2        !++
;  743     2580  2        ! All modules previously _DYING are now dead.
;  744     2581  2        !--
;  745     2582  2        MC_PTR = .PAT$GL_MC_PTR;
;  746     2583  3        WHILE( (MC_PTR = .MC_PTR [MC_NEXT]) NEQ 0 )
;  747     2584  2                DO
;  748     2585  3                BEGIN
;  749     2586  4                IF (.MC_PTR [MC_IS_DYING])
;  750     2587  3                THEN
;  751     2588  3                !++
;  752     2589  3                        ! Once _IN_RST is FALSE almost no other flags are believed.
;  753     2590  3                        !--
;  754     2591  3                        MC_PTR [MC_IN_RST] = FALSE;
```

```
;  755          2592  2           END;
;  756          2593  1 END;



                                        0FFC 00000          .ENTRY   PAT$CANC_MODULE, Save R2,R3,R4,R5,R6,R7,R8,-; 2421
                                                                     R9,R10,R11
                     5B 00000000G   EF   9E 00002          MOVAB    PAT$GL_MC_PTR, R11                           ;
                     5A      FE49   CF   9E 00009          MOVAB    PAT$GET_NXT_SAT, R10                         ;
                     59 00000000G   EF   9E 0000E          MOVAB    PAT$GL_RST_BEGN, R9                          ;
                                    56   D4 00015          CLRL     MC_CHAIN                                     ; 2464
                     55 00000000G   EF   D0 00017          MOVL     PAT$GL_HEAD_LST, POINTER                     ; 2465
                                    06   12 0001E          BNEQ     1$                                           ;
                     56             6B   D0 00020          MOVL     PAT$GL_MC_PTR, MC_CHAIN                      ; 2467
                     54             56   D0 00023          MOVL     MC_CHAIN, MC_PTR                             ;
                                    57   7C 00026 1$:      CLRQ     SHRINK_SAT                                   ; 2475
                                    56   D5 00028 2$:      TSTL     MC_CHAIN                                     ; 2482
                                    0C   13 0002A          BEQL     3$                                           ;
        50           54             69   C1 0002C          ADDL3    PAT$GL_RST_BEGN, MC_PTR, R0                  ; 2489
                     54             60   3C 00030          MOVZWL   (R0), MC_PTR                                 ;
                                    07   12 00033          BNEQ     4$                                           ;
                                  0086   31 00035          BRW      8$                                           ; 2491
                     54       04    A5   D0 00038 3$:      MOVL     4(POINTER), MC_PTR                           ; 2499
        50           54             69   C1 0003C 4$:      ADDL3    PAT$GL_RST_BEGN, MC_PTR, R0                  ; 2505
        6D      03   A0             01   E1 00040          BBC      #1, 3(R0), 6$                                ;
                                    5A   DD 00045          PUSHL    R10                                          ; 2513
                              1C    A0   9F 00047          PUSHAB   28(R0)                                       ;
           00000000V   EF           02   FB 0004A          CALLS    #2, DELE_SAT_LVT                             ;
                     57             50   C0 00051          ADDL2    R0, SHRINK_SAT                               ;
                              0F    AA   9F 00054          PUSHAB   PAT$GET_NXT_LVT                              ; 2515
        50           54             69   C1 00057          ADDL3    PAT$GL_RST_BEGN, MC_PTR, R0                  ;
                              1C    A0   9F 0005B          PUSHAB   28(R0)                                       ;
           00000000V   EF           02   FB 0005E          CALLS    #2, DELE_SAT_LVT                             ;
                     58             50   C0 00065          ADDL2    R0, SHRINK_LVT                               ;
                     50 00000000G   EF   D0 00068          MOVL     PAT$GL_CSP_PTR, R0                           ; 2523
                                    21   13 0006F          BEQL     5$                                           ;
                     51             60   D0 00071          MOVL     (R0), CS_PTR                                 ; 2535
                     53             61   9A 00074          MOVZBL   (CS_PTR), R3                                 ; 2536
        50           54             69   C1 00077          ADDL3    PAT$GL_RST_BEGN, MC_PTR, R0                  ;
                     52       0C    A0   9A 0007B          MOVZBL   12(R0), R2                                   ;
 52     00      01   A1             53   2D 0007F          CMPC5    R3, 1(CS_PTR), #0, R2, 13(R0)                ; 2537
                              0D    A0      00085                                                            ;
                                    09   12 00087          BNEQ     5$                                           ;
                                    7E   D4 00089          CLRL     -(SP)                                        ; 2539
           00000000G   EF           01   FB 0008B          CALLS    #1, PAT$SAVE_SCOPE                           ;
        50           54             69   C1 00092 5$:      ADDL3    PAT$GL_RST_BEGN, MC_PTR, R0                  ; 2547
                03   A0       40    8F   88 00096          BISB2    #64, 3(R0)                                   ;
                                    54   DD 0009B          PUSHL    MC_PTR                                       ; 2554
           00000000V   EF           01   FB 0009D          CALLS    #1, UNLINK_NT_RECS                           ;
        50           54             69   C1 000A4          ADDL3    PAT$GL_RST_BEGN, MC_PTR, R0                  ; 2555
                              1C    A0   9F 000A8          PUSHAB   28(R0)                                       ;
           00000000G   EF           01   FB 000AB          CALLS    #1, PAT$VS_FREE                              ;
                                    56   D5 000B2 6$:      TSTL     MC_CHAIN                                     ; 2560
                                    03   12 000B4          BNEQ     7$                                           ;
                     55             65   D0 000B6          MOVL     (POINTER), POINTER                           ; 2562
```

```
                                        03  13 00089 7$:    BEQL    8$                                              ;
                                      FF6A  31 0008B        BRW     2$                                              ;
                                        57  D5 0008E 8$:    TSTL    SHRINK_SAT                                      ; 2572
                                        09  13 000C0        BEQL    9$                                              ;
                                        5A  DD 000C2        PUSHL   R10                                             ; 2574
                    00000000V           01  FB 000C4        CALLS   #1, EMPTY_SAT_LVT                               ;
                                        58  D5 000CB 9$:    TSTL    SHRINK_LVT                                      ; 2575
                                        0A  13 000CD        BEQL    10$                                             ;
                                    0F  AA  9F 000CF        PUSHAB  PAT$GET_NXT_LVT                                 ; 2577
                    00000000V       EF  01  FB 000D2        CALLS   #1, EMPTY_SAT_LVT                               ;
                                    54  6B  D0 000D9 10$:   MOVL    PAT$GL_MC_PTR, MC_PTR                           ; 2582
                                50  54  69  C1 000DC 11$:   ADDL3   PAT$GL_RST_BEGN, MC_PTR, R0                     ; 2583
                                    54  60  3C 000E0        MOVZWL  (R0), MC_PTR                                    ;
                                        0F  13 000E3        BEQL    12$                                             ;
                                50  54  69  C1 000E5        ADDL3   PAT$GL_RST_BEGN, MC_PTR, R0                     ; 2586
                                EE  03  A0  06  E1 000E9    BBC     #6, 3(R0), 11$                                  ;
                                    03  A0  02  8A 000EE    BICB2   #2, 3(R0)                                       ; 2591
                                        E8  11 000F2        BRB     11$                                             ; 2583
                                            04 000F4 12$:   RET                                                     ; 2593

; Routine Size:  245 bytes,    Routine Base:  _PAT$CODE + 01AA
```

```
;  758          2594  1 ROUTINE UNLINK_NT_RECS( MC_PTR ) : NOVALUE =
;  759          2595  1
;  760          2596  1 !++
;  761          2597  1 ! Functional Description:
;  762          2598  1 !
;  763          2599  1 !       Remove all NT records for the module indicated  by MC_PTR from the
;  764          2600  1 !       hash chains they are in.  This is done in preparation for removing the
;  765          2601  1 !       indicated module from the RST completely.
;  766          2602  1 !
;  767          2603  1 ! Formal Parameters:
;  768          2604  1 !
;  769          2605  1 !       MC_PTR  -An RST pointer to the MC entry for
;  770          2606  1 !                the module we are to unlink NTs for.
;  771          2607  1 !
;  772          2608  1 ! Implicit Inputs:
;  773          2609  1 !
;  774          2610  1 !       The NT records for the indicated module are all in contiguous storage
;  775          2611  1 !       (the so-called "vector storage" for that module), and one can
;  776          2612  1 !       'go thru' this vector and pick out the NT records given the data therein.
;  777          2613  1 !       (Now we use RST_NT_OVERHEAD and the count byte for the associated symbol
;  778          2614  1 !       name - but any 'contained' method will do).
;  779          2615  1 !
;  780          2616  1 ! Implicit Outputs:
;  781          2617  1 !
;  782          2618  1 !       All existing hash chains are correctly re-linked,
;  783          2619  1 !       bypassing all NTs for the indicated module.
;  784          2620  1 !
;  785          2621  1 ! Return Value:
;  786          2622  1 !
;  787          2623  1 !       NOVALUE - we make no checks on the given data.
;  788          2624  1 !--
;  789          2625  1
;  790          2626  2 BEGIN
;  791          2627  2
;  792          2628  2 MAP
;  793          2629  2         MC_PTR : REF MC_RECORD;
;  794          2630  2
;  795          2631  2 LOCAL
;  796          2632  2         NT_PTR : REF NT_RECORD,                                 ! Pointer we use to go thru the storage
;  797          2633  2                                                                 ! vector of NT records.
;  798          2634  2         NT_VEC_DESC : REF VECT_STORE_DESC;                      ! We use the MC-contained descriptor of
;  799          2635  2                                                                 ! the vector storage for the indicated NTs.
;  800          2636  2
;  801          2637  2 !++
;  802          2638  2 ! Pick up the vector storage descriptor for the  indicated NT records.
;  803          2639  2 !--
;  804          2640  2 NT_VEC_DESC = MC_PTR [MC_NT_STORAGE];
;  805          2641  2
;  806          2642  2 !++
;  807          2643  2 ! The first NT record begins in the first byte of the storage vector, and
;  808          2644  2 ! the storage 'marker' field has been set to the first byte of unallocated
;  809          2645  2 ! storage.
;  810          2646  2 !--
;  811          2647  2 NT_PTR = .NT_VEC_DESC [STOR_BEGIN_RST];
;  812          2648  2
;  813          2649  2 !++
;  814          2650  2 ! Go thru the storage vector sequentially.
```

```
;  815          2651  2 !--
;  816          2652  3 WHILE( .NT_PTR LSSA .NT_VEC_DESC [STOR_MARKER] )
;  817          2653  2         DO                                                                          ; R
;  818          2654  3         BEGIN
;  819          2655  3         LOCAL
;  820          2656  3                 BACKWARD_NT : REF NT_RECORD,
;  821          2657  3                 FORWARD_NT  : REF NT_RECORD;
;  822          2658  3
;  823          2659  3         !++
;  824          2660  3         ! Pick up the two links (forward and backward) which indicate where in
;  825          2661  3         ! the hash chain this NT record is.
;  826          2662  3         !--
;  827          2663  3         BACKWARD_NT = .NT_PTR [NT_BACKWARD];
;  828          2664  3         FORWARD_NT = .NT_PTR [NT_FORWARD];
;  829          2665  3
;  830          2666  3         !++
;  831          2667  3         ! Unlink the NT record from its hash chain, i.e. Make the forward
;  832          2668  3         ! pointer of our back NT record point to the one after us.  Note that
;  833          2669  3         ! if our back pointer points back to the hash chain, then we are
;  834          2670  3         ! actually overwriting the RST pointer in that hash chain entry (as we
;  835          2671  3         ! want to).  This is why NT_FORWARD must be the first 2 bytes of an NT record.
;  836          2672  3         !--
;  837          2673  3         BACKWARD_NT [NT_FORWARD] = .FORWARD_NT;
;  838          2674  3
;  839          2675  3         !++
;  840          2676  3         ! If we are not at the end of the hash chain, we must connect up the
;  841          2677  3         ! hole we are making by unlinking this NT record.
;  842          2678  3         !--
;  843          2679  4         IF (.FORWARD_NT NEQ 0)
;  844          2680  3         THEN
;  845          2681  3                 FORWARD_NT [NT_BACKWARD] = .BACKWARD_NT;
;  846          2682  3
;  847          2683  3         !++
;  848          2684  3         ! Find the next NT record in the vector by adding the overhead bytes
;  849          2685  3         ! (fixed) for each record to the count bytes for the name.
;  850          2686  3         !--
;  851          2687  3         NT_PTR = .NT_PTR + RST_NT_OVERHEAD + .NT_PTR [NT_NAME_CS];
;  852          2688  2         END;
;  853          2689  2
;  854          2690  2 !++
;  855          2691  2 ! All NT records are successfully unlinked.
;  856          2692  2 !--
;  857          2693  1 END;


                                        003C 00000 UNLINK_NT_RECS:
                                                   .WORD   Save R2,R3,R4,R5                                ; 2594
                        55 00000000G    EF  D0 00002       MOVL    PAT$GL_RST_BEGN, R5                     ; 2640
             50         55        04   AC  C1 00009       ADDL3   MC_PTR, R5, R0                          ;
                        50              1C  C0 0000E       ADDL2   #28, NT_VEC_DESC                        ;
                        54        01    A0  3C 00011       MOVZWL  1(NT_VEC_DESC), NT_PTR                  ; 2647
54        05    A0      10              00  ED 00015 1$:    CMPZV   #0, #16, 5(NT_VEC_DESC), NT_PTR         ; 2652
                                        27  1B 00018       BLEQU   3$                                      ;
             53         54              55  C1 0001D       ADDL3   R5, NT_PTR, R3                          ; 2663
```

```
                52      0A    A3  3C 00021          MOVZWL  10(R3), BACKWARD_NT                      ;
                51            63  3C 00025          MOVZWL  (R3), FORWARD_NT                        ; 2664
                           6542   9F 00028          PUSHAB  (R5)[BACKWARD_NT]                       ; 2673
                9E            51  B0 0002B          MOVW    FORWARD_NT, @(SP)+                      ;
                              51  D5 0002E          TSTL    FORWARD_NT                              ; 2679
                              07  13 00030          BEQL    2$                                      ;
                         0A A541  9F 00032          PUSHAB  10(R5  FORWARD_NT]                      ; 2681
                9E            52  B0 00036          MOVW    BACKWARD_NT, @(SP)+                     ;
                51      0C    A3  9A 00039 2$:      MOVZBL  12(R3), R1                              ; 2687
                54      0D A144   9E 0003D          MOVAB   13(R1)[NT_PTR], NT_PTR                  ;
                              D1  11 00042          BRB     1$                                      ; 2652
                                  04 00044 3$:      RET                                             ; 2693

; Routine Size:  69 bytes,    Routine Base:  _PAT$CODE + 029F
```

```
;  859          2694  1 ROUTINE EMPTY_SAT_LVT( access_function ) : NOVALUE =
;  860          2695  1
;  861          2696  1 !++
;  862          2697  1 ! Functional Description:
;  863          2698  1 !
;  864          2699  1 !       Empty out the storage associated with SAT/LVT records
;  865          2700  1 !       for all modules in the MC marked MC_IS_DYING,
;  866          2701  1 !       (there may be some records therein which belong to another module)
;  867          2702  1 !       by moving any valid records therein to storage
;  868          2703  1 !       associated with a module that is not being deleted.
;  869          2704  1 !       Then release the _DYING storage since the associated modules
;  870          2705  1 !       are to be deleted from the RST.
;  871          2706  1 !
;  872          2707  1 ! Formal Parameters:
;  873          2708  1 !
;  874          2709  1 !       access_function -The address of the function we should call
;  875          2710  1 !                        to access the indicated SAT/LVT structure.
;  876          2711  1 !
;  877          2712  1 ! Implicit Inputs: (assumptions)
;  878          2713  1 !
;  879          2714  1 !       The SAT_RECORD structure is appropriate for moving SAT/LVT
;  880          2715  1 !       records around - i.e. SATs are larger than LVTs - as
;  881          2716  1 !       long as we use the record_size value returned by the
;  882          2717  1 !       SL_ACCE_INIT call instead of the one associated with
;  883          2718  1 !       the SAT record itself.
;  884          2719  1 !
;  885          2720  1 ! Implicit Outputs:
;  886          2721  1 !
;  887          2722  1 !       The indicated structure is left in the state that LOOKUP_SAT
;  888          2723  1 !       and LOOKUP_LVT assume they are in.
;  889          2724  1 !
;  890          2725  1 ! Routine Value:
;  891          2726  1 !
;  892          2727  1 !       NOVALUE
;  893          2728  1 !
;  894          2729  1 ! Side Effects:
;  895          2730  1 !
;  896          2731  1 !       The SAT/LVT records marked for deletion are overwritten
;  897          2732  1 !       by valid records; all SAT/LVT storage associated with
;  898          2733  1 !       MC_IS_DYING modules is released.
;  899          2734  1 !--
;  900          2735  1
;  901          2736  2 BEGIN
;  902          2737  2
;  903          2738  2 LOCAL
;  904          2739  2         RECORD_SIZE,                                    ! The number of bytes in each record.
;  905          2740  2         !++
;  906          2741  2         ! Declare a pointer which is suitable for use as a REF SAT_RECORD as
;  907          2742  2         ! well as a REF LVT_RECORD. This pointer is used to sequentially go thru
;  908          2743  2         ! vector storage to pick up non-deleted records. We keep two pointers
;  909          2744  2         ! to the SAT or LVT record currently under consideration.  One, 'current',
;  910          2745  2         ! is NOT a SAT_POINTER because longword pointers are not easily
;  911          2746  2         ! comparable to the NT-pointers we get from storage descriptors.  The
;  912          2747  2         ! other is a   'real' SAT_POINTER which always corresponds to 'current'.
;  913          2748  2         !--
;  914          2749  2         CURRENT,
;  915          2750  2         CURRENT_SAT_PTR : REF SAT_RECORD,
```

```
;  916          2751  2          NEXT_FREE : REF SAT_RECORD,                                  ! Corresponding to 'current' is a pointer wh
;  917          2752  2                                                                       ! where the next free SAT/LVT record is.
;  918          2753  2          STORE_DESC : REF VECT_STORE_DESC,                            ! We look at the vector storage which is all
;  919          2754  2                                                                       ! for SAT/LVTs for each _DYING module.
;  920          2755  2
;  921          2756  2          END_MARKER,                                                  ! Maintain a pointer into the vector storage
;  922          2757  2                                                                       ! says where the last+1 record associated
;  923          2758  2                                                                       ! with a given module starts.
;  924          2759  2          MC_PTR : REF MC_RECORD,                                      ! We scan the MC to pick out all _DYING modu
;  925          2760  2          MOVED_COUNT;                                                 ! Tally up how many records we move.
;  926          2761  2
;  927          2762  2  !++
;  928          2763  2  ! Set up to access the structure in the mode which we get told of FREE records
;  929          2764  2  ! only, so that we can deduce where to store the still-used ones we have to remove
;  930          2765  2  ! from the _DYING storage.  While we're at it pick up the size of the indicated
;  931          2766  2  ! fixed-size records.
;  932          2767  2  !--
;  933          2768  2  RECORD_SIZE = (.access_function)(SL_ACCE_INIT);
;  934          2769  2
;  935          2770  2  !++
;  936          2771  2  ! Scan the module chain to pick up all modules marked as _DYING.  They are the
;  937          2772  2  ! ones that we have to clean-up and release storage for. We purposely don't ever
;  938          2773  2  ! look at the MC record allocated to globals since it never _IS_DYING.
;  939          2774  2  !--
;  940          2775  2  MC_PTR = .PAT$GL_MC_PTR;
;  941          2776  3  WHILE( (MC_PTR = .MC_PTR [MC_NEXT]) NEQ 0 )
;  942          2777  2  DO
;  943          2778  3          BEGIN
;  944          2779  3          !++
;  945          2780  3          ! We only consider _DYING modules.  The _DYING flag is meaningless if
;  946          2781  3          ! the module is not _IN_RST.
;  947          2782  3          !--
;  948          2783  4          IF (.MC_PTR [MC_IN_RST] AND .MC_PTR [MC_IS_DYING])
;  949          2784  3          THEN
;  950          2785  4                  BEGIN
;  951          2786  4                  !++
;  952          2787  4                  ! For each module we have to look at the vector storage
;  953          2788  4                  ! which corresponds to the records we are to clean up.
;  954          2789  4                  !--
;  955          2790  6                  STORE_DESC = (IF (.access_function EQL PAT$GET_NXT_SAT)
;  956          2791  5                                THEN MC_PTR [MC_SAT_STORAGE]
;  957          2792  4                                ELSE MC_PTR [MC_LVT_STORAGE]);
;  958          2793  4
;  959          2794  4                  !++
;  960          2795  4                  ! The records that we want to move are those within the
;  961          2796  4                  ! bounds defined by the _DYING module that are not
;  962          2797  4                  ! already marked for deletion.  We go thru this storage
;  963          2798  4                  ! using and comparing RST-pointers, and build a corresponding
;  964          2799  4                  ! SAT-pointer when we want to use it.
;  965          2800  4                  !--
;  966          2801  4                  END_MARKER = .STORE_DESC [STOR_MARKER];
;  967          2802  5                  IF ((CURRENT = .STORE_DESC [STOR_BEGIN_RST]) EQL 0)                                  ; R
;  968          2803  4                  THEN
;  969          2804  4                          !++
;  970          2805  4                          ! No vector storage is allocated for this module
;  971          2806  4                          ! for this kind of storage.  Simply fake having
;  972          2807  4                          ! come to the end of the storage.
```

```
;  973     2808  4                    !--
;  974     2809  4                    CURRENT = .END_MARKER;
;  975     2810  5            WHILE( .CURRENT LSSA .END_MARKER )
;  976     2811  4                    DO
;  977     2812  5                    BEGIN
;  978     2813  5                    !++
;  979     2814  5                    ! Construct a SAT pointer to correspond to the
;  980     2815  5                    ! current place in the storage vector.
;  981     2816  5                    !--
;  982     2817  5                    CURRENT_SAT_PTR = .CURRENT + .PAT$GL_RST_BEGN;
;  983     2818  5
;  984     2819  5                    !++
;  985     2820  5                    ! We only want to deal with records not marked for
;  986     2821  5                    ! deletion.
;  987     2822  5                    !--
;  988     2823  6                    IF (.CURRENT_SAT_PTR [SAT_NT_PTR] NEQ 1)
;  989     2824  5                    THEN
;  990     2825  6                            BEGIN
;  991     2826  6                            !++
;  992     2827  6                            ! See if there is some place to put this record.
;  993     2828  6                            !--
;  994     2829  7                            IF ((NEXT_FREE = (.access_function)(SL_ACCE_FREE)) EQL 0)
;  995     2830  6                            THEN
;  996     2831  7                                    BEGIN
;  997     2832  7                                    !++
;  998     2833  7                                    ! This should never happen because the
;  999     2834  7                                    ! storage for a given module should be
; 1000     2835  7                                    ! big enough to contain all SATs/LVTs for
; 1001     2836  7                                    ! that module.  If this is the number of
; 1002     2837  7                                    ! records marked for deletion, any real
; 1003     2838  7                                    ! records left herein should correspond to
; 1004     2839  7                                    ! outside records now marked for deletion.
; 1005     2840  7                                    !--
; 1006     2841  7                                    $FAO_TT_OUT('!/empty free storage error');
; 1007     2842  7                                    RETURN;
; 1008     2843  6                                    END;
; 1009     2844  6
; 1010     2845  6                            !++
; 1011     2846  6                            ! Move the valid record to a safe place.
; 1012     2847  6                            !--
; 1013     2848  6                            CH$MOVE( .record_size, .CURRENT_SAT_PTR, .NEXT_FREE );
; 1014     2849  5                            END;
; 1015     2850  4
; 1016     2851  5                    !++
; 1017     2852  5                    ! Go back to look at the next record in the
; 1018     2853  5                    ! current vector storage.
; 1019     2854  5                    !--
; 1020     2855  5                    CURRENT = .CURRENT + .RECORD_SIZE;
; 1021     2856  4                    END;
; 1022     2857  4
; 1023     2858  4            !++
; 1024     2859  4            ! All records are now moved to a safe place so we
; 1025     2860  4            ! can free up the associated vector storage.
; 1026     2861  4            !--
; 1027     2862  4            PAT$VS_FREE( .STORE_DESC );
; 1028     2863  3            END;                                    ! Go back to consider the next _DYING module
; 1029     2864  2    END;                                            ! Go back and consider the next module in th
```

```
;  1030          2865  2
;  1031          2866  2 !++
;  1032          2867  2 ! All SAT/LVT storage has been freed up - we still have to leave what's left in
;  1033          2868  2 ! order so that the accessing functions can use them.  (LOOKUP_SAT/LVT)
;  1034          2869  2 ! PAT$SORT_SA_LVT believes MC_IN_RST which is at this point untrue, but since
;  1035          2870  2 ! the associated storage descriptor has been zeroed out, the accessing function
;  1036          2871  2 ! will work OK.
;  1037          2872  2 !--
;  1038          2873  2 PAT$SORT_SA_LVT( .access_function );
;  1039          2874  2
;  1040          2875  1 END;
;  INFO#212                 L1:2841
;  Null expression appears in value-required context


                                                        .PSECT  _PAT$PLIT,NOWRT,NOEXE,0

                                                    1A  00000 P.AAA:  .BYTE   26                                                ;
74  73  20  65  65  72  66  20  79  74  70  6D  65  2F  21  00001        .ASCII  \!/empty free storage error\                        ;
                    72  6F  72  72  65  20  65  67  61  72  6F  00010                                                            ;


                                                        .PSECT  _PAT$CODE,NOWRT,2

                                                  0FFC 00000 EMPTY_SAT_LVT:
                                                             .WORD   Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11                       ; 2694
                                        5E              04   C2 00002          SUBL2   #4, SP                                      ;
                                                        7E   D4 00005          CLRL    -(SP)                                       ; 2768
                                    04  BC              01   FB 00007          CALLS   #1, @ACCESS_FUNCTION                        ;
                                        6E              50   D0 0000B          MOVL    R0, RECORD_SIZE                             ;
                                        57  00000000G   EF   D0 0000E          MOVL    PAT$GL_MC_PTR, MC_PTR                       ; 2775
50                                      57  00000000G   EF   C1 00015 1$:      ADDL3   PAT$GL_RST_BEGN, MC_PTR, R0                 ; 2776
                                        57              60   3C 0001D          MOVZWL  (R0), MC_PTR                                ;
                                                        76   13 00020          BEQL    8$                                          ;
50                                      57  00000000G   EF   C1 00022          ADDL3   PAT$GL_RST_BEGN, MC_PTR, R0                 ; 2783
E6                  03  A0              01   E1 0002A          BBC     #1, 3(R0), 1$                                       ;
E1                  03  A0              06   E1 0002F          BBC     #6, 3(R0), 1$                                       ;
                        51      FCE4    CF   9E 00034          MOVAB   PAT$GET_NXT_SAT, R1                                 ; 2790
                        51        04    AC   D1 00039          CMPL    ACCESS_FUNCTION, R1                                 ;
                                        06   12 0003D          BNEQ    2$                                                  ;
                        56        23    A0   9E 0003F          MOVAB   35(R0), STORE_DESC                                  ; 2791
                                        04   11 00043          BRB     3$                                                  ;
                        56        2A    A0   9E 00045 2$:      MOVAB   42(R0), STORE_DESC                                  ; 2792
                        59        05    A6   3C 00049 3$:      MOVZWL  5(STORE_DESC), END_MARKER                           ; 2801
                        58        01    A6   3C 0004D          MOVZWL  1(STORE_DESC), CURRENT                              ; 2802
                                        03   12 00051          BNEQ    4$                                                  ;
                        58              59   D0 00053          MOVL    END_MARKER, CURRENT                                 ; 2809
                        59              58   D1 00056 4$:      CMPL    CURRENT, END_MARKER                                 ; 2810
                                        31   1E 00059          BGEQU   7$                                                  ;
5A                      58  00000000G   EF   C1 0005B          ADDL3   PAT$GL_RST_BEGN, CURRENT, CURRENT_SAT_PTR           ; 2817
                        01              6A   B1 00063          CMPW    (CURRENT_SAT_PTR), #1                               ; 2823
                                        1F   13 00066          BEQL    6$                                                  ;
                                        03   DD 00068          PUSHL   #3                                                  ; 2829
                    04  BC              01   FB 0006A          CALLS   #1, @ACCESS_FUNCTION                                ;
                        5B              50   D0 0006E          MOVL    R0, NEXT_FREE                                       ;
```

```
                                    10  12 00071          BNEQ    5$                                                          ;
                                    7E  D4 00073          CLRL    -(SP)                                                       ; 2841
                         00000000'  EF  9F 00075          PUSHAB  P.AAA                                                       ;
                00000000G   EF      02  FB 0007B          CALLS   #2, PAT$FAO_OUT                                             ;
                                        04 00082          RET                                                                 ; 2831
            6B          6A          6E  28 00083 5$:      MOVC3   RECORD_SIZE, (CURRENT_SAT_PTR), (NEXT_FREE)                 ; 2848
                        58          6E  C0 00087 6$:      ADDL2   RECORD_SIZE, CURRENT                                        ; 2855
                                    CA  11 0008A          BRB     4$                                                          ; 2810
                                    56  DD 0008C 7$:      PUSHL   STORE_DESC                                                  ; 2862
                00000000G   EF      01  FB 0008E          CALLS   #1, PAT$VS_FREE                                             ;
                                FF7D    31 00095          BRW     1$                                                          ; 2776
                             04     AC  DD 00098 8$:      PUSHL   ACCESS_FUNCTION                                             ; 2873
                00000000V   EF      01  FB 0009B          CALLS   #1, PAT$SORT_SA_LVT                                         ;
                                        04 000A2          RET                                                                 ; 2875

; Routine Size:  163 bytes,    Routine Base:  _PAT$CODE + 02E4
```

```
;1042     2876  1 GLOBAL ROUTINE PAT$SORT_SA_LVT( ACCESS_FUNCTION ) : NOVALUE =
;1043     2877  1
;1044     2878  1 !++
;1045     2879  1 ! Functional Description:
;1046     2880  1 !
;1047     2881  1 !       This routine is an unspeakably inefficient, (shell) sort
;1048     2882  1 !       suitable to sort either the LVT or the SAT into ascending order.
;1049     2883  1 !       It is necessary because access to the LVT or SAT assumes that
;1050     2884  1 !       it has been done.
;1051     2885  1 !
;1052     2886  1 ! Formal Parameters:
;1053     2887  1 !
;1054     2888  1 !       ACCESS_FUNCTION -The function which must be called to
;1055     2889  1 !                        gain sequential access to the SAT or LVT.
;1056     2890  1 !
;1057     2891  1 ! Implicit Inputs: (assumptions)
;1058     2892  1 !
;1059     2893  1 !       SATs/LVTs are accessed via longword pointers, vector
;1060     2894  1 !       storage descriptors contain RST-pointers, and CH$MOVE
;1061     2895  1 !       works with the same kind of pointer as SAT_POINTER does.
;1062     2896  1 !
;1063     2897  1 !       The SAT_RECORD structure is appropriate for either type of
;1064     2898  1 !       vector providing that we only use the _NT_PTR and _LB fields.
;1065     2899  1 !
;1066     2900  1 !       A temporary SAT record is large enough to contain an LVT record.
;1067     2901  1 !
;1068     2902  1 !       The sort is on the _LB field, which corresponds to the _VALUE
;1069     2903  1 !       field in LVT records.  The comparison is UNSIGNED.
;1070     2904  1 !
;1071     2905  1 ! Implicit Outputs:
;1072     2906  1 !
;1073     2907  1 !       None.
;1074     2908  1 !
;1075     2909  1 ! Routine Value:
;1076     2910  1 !
;1077     2911  1 !       NOVALUE
;1078     2912  1 !
;1079     2913  1 ! Side Effects:
;1080     2914  1 !
;1081     2915  1 !       The SAT/LVT vector is sorted into ascending order.
;1082     2916  1 !
;1083     2917  1 !       This routine knows the difference between RST pointers and 'real'
;1084     2918  1 !       pointers.  (The former are stored in VECT_STORE_DESCS and the latter
;1085     2919  1 !       are used to reference SAT/LVT structures.)
;1086     2920  1 !--
;1087     2921  1
;1088     2922  2 BEGIN
;1089     2923  2
;1090     2924  2 LABEL
;1091     2925  2         INIT_LOOP,
;1092     2926  2         SORT_LOOP;
;1093     2927  2
;1094     2928  2 LOCAL
;1095     2929  2         MC_PTR : REF MC_RECORD,                         ! Current module pointer
;1096     2930  2         STORE_DESC : REF VECT_STORE_DESC,               ! Storage within the current module
;1097     2931  2         NXT_MC_PTR : REF MC_RECORD,                     ! Pointer to next module
;1098     2932  2         NXT_STORE_DESC : REF VECT_STORE_DESC,           ! Storage within the next module
```

```
;  1099     2933  2          SAT_FLAG,                                       ! 1=SAT, 0=LVT SORT
;  1100     2934  2          ELEMENT_SIZE,                                   ! The number of bytes in a SAT/LVT record.
;  1101     2935  2          REC_PTR : REF SAT_RECORD;                       ! Declare a pointer which is suitable for us
;  1102     2936  2                                                          ! a REF SAT_RECORD as well as a REF LVT_RECO
;  1103     2937  2                                                          ! This pointer is used to sequentially go th
;  1104     2938  2                                                          ! the indicated vector to do the sort.
;  1105     2939  2
;  1106     2940  2      !++
;  1107     2941  2      ! The (shell) sort works by going thru the vector sequentially, considering
;  1108     2942  2      ! each element, finding the minimum of those which are left, and swapping
;  1109     2943  2      ! this minimum with the current one if such a minimum can be found.
;  1110     2944  2      !
;  1111     2945  2      ! REC_PTR points to the record currently under consideration.
;  1112     2946  2      ! MIN_PTR points to the smallest one of those left.
;  1113     2947  2      ! TMP_PTR is used to deduce what MIN_PTR should be.
;  1114     2948  2      !
;  1115     2949  2      ! Set up to begin accessing the LVT or SAT.
;  1116     2950  2      !--
;  1117     2951  2      ELEMENT_SIZE = (.ACCESS_FUNCTION)( SL_ACCE_INIT );
;  1118     2952  2      SAT_FLAG = .ACCESS_FUNCTION EQL PAT$GET_NXT_SAT;
;  1119     2953  2      MC_PTR = .PAT$GL_MC_PTR;
;  1120     2954  2      REPEAT
;  1121     2955  2          INIT_LOOP:
;  1122     2956  3              BEGIN
;  1123     2957  3              !++
;  1124     2958  3              !  Check for the end of the module chain.
;  1125     2959  3              !--
;  1126     2960  4              IF (.MC_PTR EQL 0)
;  1127     2961  3              THEN
;  1128     2962  3                      !++
;  1129     2963  3                      ! The SAT/LVT structure is completely empty.  This is not a
;  1130     2964  3                      ! sort error.
;  1131     2965  3                      !--
;  1132     2966  3                      RETURN(TRUE);
;  1133     2967  3
;  1134     2968  3              !++
;  1135     2969  3              ! The current module is to be ignored if the user has not brought it
;  1136     2970  3              ! into the RST.
;  1137     2971  3              !--
;  1138     2972  3              IF NOT .MC_PTR[MC_IN_RST]
;  1139     2973  3              THEN
;  1140     2974  4                      BEGIN
;  1141     2975  4                      MC_PTR = .MC_PTR[MC_NEXT];
;  1142     2976  4                      LEAVE INIT_LOOP;
;  1143     2977  3                      END;
;  1144     2978  3
;  1145     2979  3              !++
;  1146     2980  3              ! Check if this module has any of the kind of storage sought.
;  1147     2981  3              ! No storage is allocated if the "BEGIN" field is null.
;  1148     2982  3              !--
;  1149     2983  4              STORE_DESC = (IF .SAT_FLAG
;  1150     2984  4                            THEN MC_PTR[MC_SAT_STORAGE]
;  1151     2985  3                            ELSE MC_PTR[MC_LVT_STORAGE]);
;  1152     2986  4              IF (.STORE_DESC[STOR_BEGIN_RST] EQL 0)
;  1153     2987  3              THEN
;  1154     2988  4                      BEGIN
;  1155     2989  4                      MC_PTR = .MC_PTR[MC_NEXT];
```

```
;  1156          2990  4                   LEAVE INIT_LOOP;
;  1157          2991  4                   END;
;  1158          2992  3
;  1159          2993  3           !++
;  1160          2994  3           ! The first SAT/LVT record has been found.  Discontinue looping.
;  1161          2995  3           !--
;  1162          2996  3           REC_PTR = .STORE_DESC[STOR_BEGIN_RST] + .PAT$GL_RST_BEGN;
;  1163          2997  3           EXITLOOP;
;  1164          2998  3           END;
;  1165          2999  2 !++
;  1166          3000  2 ! There is at least one record in the logical SAT/LVT structure.  Now loop
;  1167          3001  2 ! to actually do the sort.
;  1168          3002  2 !--
;  1169          3003  2 REPEAT
;  1170          3004  3           BEGIN
;  1171          3005  3           LABEL
;  1172          3006  3                   MIN_LOOP;
;  1173          3007  3           LOCAL
;  1174          3008  3                   NXT_STORE : REF VECT_STORE_DESC,
;  1175          3009  3                   TMP_STORE : REF VECT_STORE_DESC,
;  1176          3010  3                   FIRST_FLAG,
;  1177          3011  3                   NXT_MC_PTR : REF MC_RECORD,
;  1178          3012  3                   TMP_MC_PTR : REF MC_RECORD,
;  1179          3013  3                   MIN_PTR : REF SAT_RECORD,
;  1180          3014  3                   NXT_PTR,
;  1181          3015  3                   TMP_PTR;
;  1182          3016  3           TMP_PTR = .REC_PTR - .PAT$GL_RST_BEGN,
;  1183          3017  3           MIN_PTR = .REC_PTR;
;  1184          3018  3           !++
;  1185          3019  3           ! Find the minimum of those left.
;  1186          3020  3           !--
;  1187          3021  3           FIRST_FLAG = TRUE;
;  1188          3022  3           TMP_MC_PTR = .MC_PTR;
;  1189          3023  3           TMP_STORE = .STORE_DESC;
;  1190          3024  3
;  1191          3025  3           REPEAT
;  1192          3026  3               MIN_LOOP:
;  1193          3027  4                   BEGIN
;  1194          3028  4                   LOCAL
;  1195          3029  4                           TMP_SAT_PTR : REF SAT_RECORD;
;  1196          3030  4
;  1197          3031  5                   IF (.TMP_PTR EQL 0)
;  1198          3032  4                   THEN
;  1199          3033  5                           BEGIN
;  1200          3034  5                           !++
;  1201          3035  5                           ! The next record must come from the next module in
;  1202          3036  5                           ! the chain.
;  1203          3037  5                           !--
;  1204          3038  6                           IF ((TMP_MC_PTR = .TMP_MC_PTR[MC_NEXT]) EQL 0)
;  1205          3039  5                           THEN
;  1206          3040  5                                   EXITLOOP;                       ! No more for min search
;  1207          3041  5                           IF NOT .TMP_MC_PTR[MC_IN_RST]
;  1208          3042  5                           THEN
;  1209          3043  5                                   LEAVE MIN_LOOP;
;  1210          3044  5
;  1211          3045  5                           !++
;  1212          3046  5                           ! Check if this module has any of the kind of storage
```

```
;1213    3047  5                ! sought.  No storage is allocated if the "BEGIN" field
;1214    3048  5                ! is null.
;1215    3049  5                !--
;1216    3050  6                TMP_STORE = (IF .SAT_FLAG
;1217    3051  6                             THEN TMP_MC_PTR[MC_SAT_STORAGE]
;1218    3052  5                             ELSE TMP_MC_PTR[MC_LVT_STORAGE]);
;1219    3053  6                IF (.TMP_STORE[STOR_BEGIN_RST] EQL 0)
;1220    3054  5                THEN
;1221    3055  5                        LEAVE MIN_LOOP;
;1222    3056  5                TMP_PTR = .TMP_STORE[STOR_BEGIN_RST];
;1223    3057  5                IF .FIRST_FLAG
;1224    3058  5                THEN
;1225    3059  6                        BEGIN
;1226    3060  6                        NXT_PTR = 0;
;1227    3061  6                        NXT_STORE = .TMP_STORE;
;1228    3062  6                        NXT_MC_PTR = .TMP_MC_PTR;
;1229    3063  6                        END
;1230    3064  5                END
;1231    3065  4        ELSE
;1232    3066  5                IF NOT ((TMP_PTR = .TMP_PTR + .ELEMENT_SIZE) LSSA
;1233    3067  5                        (.TMP_STORE[STOR_MARKER]))
;1234    3068  4                THEN
;1235    3069  5                        BEGIN
;1236    3070  5                        TMP_PTR = 0;
;1237    3071  5                        LEAVE MIN_LOOP;
;1238    3072  5                        END
;1239    3073  4                ELSE
;1240    3074  4                        IF .FIRST_FLAG
;1241    3075  4                        THEN
;1242    3076  4                                NXT_PTR = .TMP_PTR;
;1243    3077  4                !++
;1244    3078  4                ! Check if a new minimum has been found.
;1245    3079  4                !--
;1246    3080  4                TMP_SAT_PTR = .TMP_PTR + .PAT$GL_RST_BEGN;
;1247    3081  5                IF (.TMP_SAT_PTR[SAT_LB] LSSA .MIN_PTR[SAT_LB])
;1248    3082  4                THEN
;1249    3083  4                        MIN_PTR = .TMP_SAT_PTR; ! A NEW MINIMUM WAS FOUND
;1250    3084  4                FIRST_FLAG = FALSE;
;1251    3085  3                END;                                    ! End of MIN_LOOP
;1252    3086  3        !++
;1253    3087  3        ! If no "NEXT" pointer was found above, there were no more
;1254    3088  3        ! records to search.
;1255    3089  3        !--
;1256    3090  3        IF .FIRST_FLAG
;1257    3091  3        THEN
;1258    3092  3                EXITLOOP;
;1259    3093  3
;1260    3094  3    !++
;1261    3095  3    ! If the minimum has not changed, having started out as the current
;1262    3096  3    ! one, then we do nothing except go on to the next element in the
;1263    3097  3    ! vector.  Otherwise we swap the minimum with the current so that we
;1264    3098  3    ! can then go on to the next anyway.
;1265    3099  3    !--
;1266    3100  4    IF (.MIN_PTR NEQ .REC_PTR)
;1267    3101  3    THEN
;1268    3102  4            BEGIN
;1269    3103  4            LOCAL
```

```
;  1270          3104  4                       TMP_RECORD : SAT_RECORD;
;  1271          3105  4
;  1272          3106  4                   !++
;  1273          3107  4                   ! Swap the old current one with the new minimum.
;  1274          3108  4                   !--
;  1275          3109  4                   CH$MOVE( .ELEMENT_SIZE, .REC_PTR, TMP_RECORD );
;  1276          3110  4                   CH$MOVE( .ELEMENT_SIZE, .MIN_PTR, .REC_PTR );
;  1277          3111  4                   CH$MOVE( .ELEMENT_SIZE, TMP_RECORD, .MIN_PTR );
;  1278          3112  3                   END;
;  1279          3113  3
;  1280          3114  3           !++
;  1281          3115  3           ! Update the "CURRENT" record by effectively going back to the first
;  1282          3116  3           ! "NEXT" record found above.
;  1283          3117  3           !--
;  1284          3118  4           IF ((REC_PTR = .NXT_PTR) EQL 0)
;  1285          3119  3           THEN
;  1286          3120  4                   BEGIN
;  1287          3121  4                   MC_PTR = .NXT_MC_PTR;
;  1288          3122  4                   STORE_DESC = .NXT_STORE;
;  1289          3123  4                   REC_PTR = .STORE_DESC[STOR_BEGIN_RST];
;  1290          3124  3                   END;
;  1291          3125  3           REC_PTR = .REC_PTR + .PAT$GL_RST_BEGN;
;  1292          3126  2           END;                                            ! End of outer loop
;  1293          3127  2
;  1294          3128  2 !++
;  1295          3129  2 ! At this point the vector is sorted into ascending order.
;  1296          3130  2 !--
;  1297          3131  2 RETURN(TRUE);
;  1298          3132  1 END;

                                    0FFC 00000          .ENTRY  PAT$SORT_SA_LVT, Save R2,R3,R4,R5,R6,R7,R8,-; 2876
                                                                R9,R10,R11
                          5E              1C  C2 00002          SUBL2   #28, SP
                                          7E  D4 00005          CLRL    -(SP)                                 ; 2951
                    04    BC              01  FB 00007          CALLS   #1, @ACCESS_FUNCTION
                    04    AE              50  D0 0000B          MOVL    R0, ELEMENT_SIZE
                          51      FC66    CF  9E 0000F          MOVAB   PAT$GET_NXT_SAT, R1                   ; 2952
                                          50  D4 00014          CLRL    R0
                          51        04    AC  D1 00016          CMPL    ACCESS_FUNCTION, R1
                                          02  12 0001A          BNEQ    1$
                                          50  D6 0001C          INCL    R0
                          6E              50  D0 0001E 1$:      MOVL    R0, SAT_FLAG
                          57 00000000G    EF  D0 00021          MOVL    PAT$GL_MC_PTR, MC_PTR                 ; 2953
                                          01  12 00028 2$:      BNEQ    3$                                    ; 2960
                                              04 0002A          RET
                          51 00000000G    EF  D0 0002B 3$:      MOVL    PAT$GL_RST_BEGN, R1                   ; 2972
                50        57              51  C1 00032          ADDL3   R1, MC_PTR, R0
                1A  03    A0              01  E1 00036          BBC     #1, 3(R0), 6$
                          0A              6E  E9 0003B          BLBC    SAT_FLAG, 4$                          ; 2983
                50        57              51  C1 0003E          ADDL3   R1, MC_PTR, R0                        ; 2984
                          56        23    A0  9E 00042          MOVAB   35(R0), STORE_DESC
                                          08  11 00046          BRB     5$
                50        57              51  C1 00048 4$:      ADDL3   R1, MC_PTR, R0                        ; 2985
```

```
                                56        2A    A0  9E 0004C          MOVAB   42(R0), STORE_DESC                       ;
                                          01    A6  B5 00050 5$:       TSTW    1(STORE_DESC)                           ; 2986
                                                05  12 00053          BNEQ    7$
                                57              60  3C 00055 6$:       MOVZWL  (R0), MC_PTR                            ; 2989
                                                CE  11 00058          BRB     2$                                      ; 2990
                                5A        01    A6  3C 0005A 7$:       MOVZWL  1(STORE_DESC), REC_PTR                  ; 2996
                                5A              51  C0 0005E          ADDL2   R1, REC_PTR
                                59 00000000G    EF  D0 00061          MOVL    PAT$GL_RST_BEGN, R9                     ; 3016
                          53    5A              59  C3 00068 8$:       SUBL3   R9, REC_PTR, TMP_PTR
                                58              5A  D0 0006C          MOVL    REC_PTR, MIN_PTR                        ; 3017
                                54              01  D0 0006F          MOVL    #1, FIRST_FLAG                          ; 3021
                                50              56  7D 00072          MOVQ    STORE_DESC, TMP_STORE                   ; 3023
                                                53  D5 00075 9$:       TSTL    TMP_PTR                                 ; 3031
                                                36  12 00077          BNEQ    12$
                                              6941  9F 00079          PUSHAB  (R9)[TMP_MC_PTR]                        ; 3038
                                51              9E  3C 0007C          MOVZWL  @(SP)+, TMP_MC_PTR
                                                56  13 0007F          BEQL    16$
                          52    51              59  C1 00081          ADDL3   R9, TMP_MC_PTR, R2                      ; 3041
                          EB    03    A2          01  E1 00085          BBC     #1, 3(R2), 9$
                                                06  E9 0008A          BLBC    SAT_FLAG, 10$                           ; 3050
                                50        23    A2  9E 0008D          MOVAB   35(R2), TMP_STORE                       ; 3051
                                                04  11 00091          BRB     11$
                                50        2A    A2  9E 00093 10$:      MOVAB   42(R2), TMP_STORE                       ; 3052
                                          01    A0  B5 00097 11$:      TSTW    1(TMP_STORE)                            ; 3053
                                                D9  13 0009A          BEQL    9$
                                53        01    A0  3C 0009C          MOVZWL  1(TMP_STORE), TMP_PTR                   ; 3056
                                22              54  E9 000A0          BLBC    FIRST_FLAG, 14$                         ; 3057
                                                5B  D4 000A3          CLRL    NXT_PTR                                 ; 3060
                                      0C    AE        50  D0 000A5          MOVL    TMP_STORE, NXT_STORE                    ; 3061
                                      08    AE        51  D0 000A9          MOVL    TMP_MC_PTR, NXT_MC_PTR                  ; 3062
                                                      16  11 000AD          BRB     14$                                     ; 3057
                                53        04    AE  C0 000AF 12$:      ADDL2   ELEMENT_SIZE, TMP_PTR                   ; 3066
                    53    05    A0              10  00 ED 000B3          CMPZV   #0, #16, 5(TMP_STORE), TMP_PTR          ; 3067
                                                04  1A 000B9          BGTRU   13$
                                                53  D4 000BB          CLRL    TMP_PTR                                 ; 3070
                                                B6  11 000BD          BRB     9$                                      ; 3071
                                03              54  E9 000BF 13$:      BLBC    FIRST_FLAG, 14$                         ; 3074
                                5B              53  D0 000C2          MOVL    TMP_PTR, NXT_PTR                        ; 3076
                          52    53              59  C1 000C5 14$:      ADDL3   R9, TMP_PTR, TMP_SAT_PTR                ; 3080
                                      02    A8        02  A2 D1 000C9          CMPL    2(TMP_SAT_PTR), 2(MIN_PTR)              ; 3081
                                                      03  1E 000CE          BGEQU   15$
                                58              52  D0 000D0          MOVL    TMP_SAT_PTR, MIN_PTR                    ; 3083
                                                54  D4 000D3 15$:      CLRL    FIRST_FLAG                              ; 3084
                                                9E  11 000D5          BRB     9$                                      ; 3023
                                2D              54  E8 000D7 16$:      BLBS    FIRST_FLAG, 19$                         ; 3090
                                5A              58  D1 000DA          CMPL    MIN_PTR, REC_PTR                        ; 3100
                                                11  13 000DD          BEQL    17$
                          10    AE        6A        04  AE 28 000DF          MOVC3   ELEMENT_SIZE, (REC_PTR), TMP_RECORD     ; 3109
                                6A        68        04  AE 28 000E5          MOVC3   ELEMENT_SIZE, (MIN_PTR), (REC_PTR)      ; 3110
                                68        10    AE  04  AE 28 000EA          MOVC3   ELEMENT_SIZE, TMP_RECORD, (MIN_PTR)     ; 3111
                                5A              5B  D0 000F0 17$:      MOVL    NXT_PTR, REC_PTR                        ; 3118
                                                0C  12 000F3          BNEQ    18$
                                57        08    AE  D0 000F5          MOVL    NXT_MC_PTR, MC_PTR                      ; 3121
                                56        0C    AE  D0 000F9          MOVL    NXT_STORE, STORE_DESC                   ; 3122
                                5A        01    A6  3C 000FD          MOVZWL  1(STORE_DESC), REC_PTR                  ; 3123
                                5A              59  C0 00101 18$:      ADDL2   R9, REC_PTR                             ; 3125
                                              FF61  31 00104          BRW     8$                                      ; 2998
```

```
                04 00107 19$:     RET                                                   ; 3132

; Routine Size:  264 bytes,    Routine Base:  _PAT$CODE + 0387
```

```
;  1300     3133  1 ROUTINE DELE_SAT_LVT( STORE_DESC_ADDR, ACCESS_FUNCTION ) =
;  1301     3134  1
;  1302     3135  1 !++
;  1303     3136  1 ! Functional Description:
;  1304     3137  1 !
;  1305     3138  1 !       This routine deletes records from the LVT or SAT
;  1306     3139  1 !       vector it is given a pointer to (VEC_PTR).  The
;  1307     3140  1 !       records that are deleted are those that correspond
;  1308     3141  1 !       (point) to an indicated module (STORE_DESC_ADDR).
;  1309     3142  1 !       The correspondence is discovered by seeing if the
;  1310     3143  1 !       NT_PTR in a given LVT/SAT record lies within the
;  1311     3144  1 !       RST storage for the NTs for the indicated module.
;  1312     3145  1 !       This is faster than actually following the scope
;  1313     3146  1 !       chains of the pointed-to NT records to see if
;  1314     3147  1 !       they belong to the indicated module.
;  1315     3148  1 !
;  1316     3149  1 ! Formal Parameters:
;  1317     3150  1 !
;  1318     3151  1 !       STORE_DESC_ADDR -The address of the storage vector descriptor
;  1319     3152  1 !                        which completely describes the NT storage
;  1320     3153  1 !                        for the indicated module.
;  1321     3154  1 !       ACCESS_FUNCTION -The name of the function to call to
;  1322     3155  1 !                        access the indicated structure.
;  1323     3156  1 !
;  1324     3157  1 ! Implicit Inputs: (assumptions)
;  1325     3158  1 !
;  1326     3159  1 !       We assume that direct comparisons of RST pointers
;  1327     3160  1 !       is valid.  (i.e. that "begin < ptr < end" is a valid
;  1328     3161  1 !       test of whether 'ptr' RST-points to somewhere inbetween
;  1329     3162  1 !       where 'begin' and 'end' RST-point.
;  1330     3163  1 !
;  1331     3164  1 ! Implicit Outputs:
;  1332     3165  1 !
;  1333     3166  1 !       None.
;  1334     3167  1 !
;  1335     3168  1 ! Routine Value:
;  1336     3169  1 !
;  1337     3170  1 !       The number of records which are marked for deletion
;  1338     3171  1 !       in the indicated structure.
;  1339     3172  1 !
;  1340     3173  1 ! Side Effects:
;  1341     3174  1 !
;  1342     3175  1 !       Records are 'deleted' from the SAT/LVT vector.
;  1343     3176  1 !       Really, this only means that the indicated records
;  1344     3177  1 !       are marked with their NT_PTR = 1.  More processing
;  1345     3178  1 !       of this structure is necessary before it is usable,
;  1346     3179  1 !       but it is better to do this processing only once
;  1347     3180  1 !       if several modules are to be deleted from the RST.
;  1348     3181  1 !--
;  1349     3182  1
;  1350     3183  2 BEGIN
;  1351     3184  2
;  1352     3185  2 MAP
;  1353     3186  2         STORE_DESC_ADDR : REF VECT_STORE_DESC;                  ! Storage descriptor for the NTs.
;  1354     3187  2
;  1355     3188  2 LOCAL
;  1356     3189  2         GLOBALS : REF VECT_STORE_DESC,                          ! We pick up a storage descriptor for GLOBAL
```

```
;  1357     3190  2          DELETE_COUNT,                                   ! Tally up how many records we delete.
;  1358     3191  2          REC_PTR : REF SAT_RECORD,                       ! Declare a pointer which is suitable for us
;  1359     3192  2                                                          ! a REF SAT_RECORD as well as a REF LVT_RECO
;  1360     3193  2                                                          ! This pointer is used to sequentially go th
;  1361     3194  2                                                          ! the indicated vector.
;  1362     3195  2          NT_BEGINS : REF NT_RECORD,                      ! NT storage RST limits.
;  1363     3196  2          NT_ENDS   : REF NT_RECORD,
;  1364     3197  2          GL_NTS_BEGIN : REF NT_RECORD,
;  1365     3198  2          GL_NTS_END   : REF NT_RECORD;
;  1366     3199  2
;  1367     3200  2  !++
;  1368     3201  2  ! We tally up the number of records which we mark
;  1369     3202  2  ! for deletion so that we can return this value.
;  1370     3203  2  !--
;  1371     3204  2  DELETE_COUNT = 0;
;  1372     3205  2
;  1373     3206  2  !++
;  1374     3207  2  ! Pick up the RST limit pointers of where the associated NT records exist,
;  1375     3208  2  ! and where the  global NT records exist.
;  1376     3209  2  !--
;  1377     3210  2  GLOBALS = PAT$GL_MC_PTR [MC_NT_STORAGE];
;  1378     3211  2  NT_BEGINS = .STORE_DESC_ADDR [STOR_BEGIN_RST];
;  1379     3212  2  NT_ENDS = .STORE_DESC_ADDR [STOR_END_RST];
;  1380     3213  2  GL_NTS_BEGIN = .GLOBALS [STOR_BEGIN_RST];
;  1381     3214  2  GL_NTS_END = .GLOBALS [STOR_END_RST];
;  1382     3215  2
;  1383     3216  2  !++
;  1384     3217  2  ! Go thru the structure sequentially checking for records to mark as deleted.
;  1385     3218  2  !--
;  1386     3219  2  (.ACCESS_FUNCTION)( SL_ACCE_INIT );
;  1387     3220  3  WHILE( (REC_PTR = (.ACCESS_FUNCTION)( SL_ACCE_RECS )) NEQ 0 )
;  1388     3221  2  DO
;  1389     3222  3          BEGIN
;  1390     3223  3          !++
;  1391     3224  3          ! Delete the record if it points to an NT record
;  1392     3225  3          ! in the storage for the indicated module.
;  1393     3226  3          !--
;  1394     3227  3          IF (.REC_PTR [SAT_NT_PTR] GEQA .NT_BEGINS) AND
;  1395     3228  4             (.REC_PTR [SAT_NT_PTR] LSSA .NT_ENDS)
;  1396     3229  3          THEN
;  1397     3230  4                  BEGIN
;  1398     3231  4                  !++
;  1399     3232  4                  ! Actually, we can only mark it for deletion because it is much
;  1400     3233  4                  ! more efficient to compress the entire structure only once.
;  1401     3234  4                  !--
;  1402     3235  4                  DELETE_COUNT = .DELETE_COUNT +1;
;  1403     3236  4                  REC_PTR [SAT_NT_PTR] = 1;
;  1404     3237  4                  END
;  1405     3238  3          ELSE
;  1406     3239  3          !++
;  1407     3240  3          ! If that didn't detect something to delete, we must also see if we are
;  1408     3241  3          ! deleting global literals since they must go away too.
;  1409     3242  3          !--
;  1410     3243  4          IF (.ACCESS_FUNCTION EQL PAT$GET_NXT_LVT)
;  1411     3244  3          THEN
;  1412     3245  3                  IF (.REC_PTR [SAT_NT_PTR] GEQA .GL_NTS_BEGIN) AND
;  1413     3246  4                      (.REC_PTR [SAT_NT_PTR] LSSA .GL_NTS_END)
```

```
;  1414     3247  3                    THEN
;  1415     3248  4                        BEGIN
;  1416     3249  4                        DELETE_COUNT = .DELETE_COUNT +1;
;  1417     3250  4                        REC_PTR [SAT_NT_PTR] = 1;
;  1418     3251  3                        END;
;  1419     3252  2                    END;                                    ! Loop back to consider the next record.
;  1420     3253  2
;  1421     3254  2 !++
;  1422     3255  2
;  1423     3256  2 ! Return the number of records which we  have marked for deletion.
;  1424     3257  2 !--
;  1425     3258  2 RETURN(.DELETE_COUNT);
;  1426     3259  1 END;


                                        007C 00000 DELE_SAT_LVT:
                                                   .WORD   Save R2,R3,R4,R5,R6                                 ; 3133
                                             52 D4 00002          CLRL    DELETE_COUNT                          ; 3204
50 00000000G  EF 00000000G                   EF C1 00004          ADDL3   PAT$GL_RST_BEGN, PAT$GL_MC_PTR, R0    ; 3210
                        50                   1C C0 00010          ADDL2   #28, GLOBALS                          ;
                        51          04       AC D0 00013          MOVL    STORE_DESC_ADDR, R1                   ; 3211
                        56          01       A1 3C 00017          MOVZWL  1(R1), NT_BEGINS                      ;
                        55          03       A1 3C 0001B          MOVZWL  3(R1), NT_ENDS                        ; 3212
                        54          01       A0 3C 0001F          MOVZWL  1(GLOBALS), GL_NTS_BEGIN              ; 3213
                        53          03       A0 3C 00023          MOVZWL  3(GLOBALS), GL_NTS_END                ; 3214
                                             7E D4 00027          CLRL    -(SP)                                 ; 3219
                    08  BC                   01 FB 00029          CALLS   #1, @ACCESS_FUNCTION                  ;
                                             01 DD 0002D 1$:      PUSHL   #1                                    ; 3220
                    08  BC                   01 FB 0002F          CALLS   #1, @ACCESS_FUNCTION                  ;
                                             50 D5 00033          TSTL    REC_PTR                               ;
                                             2E 13 00035          BEQL    4$                                    ;
        56          60          10           00 ED 00037          CMPZV   #0, #16, (REC_PTR), NT_BEGINS         ; 3227
                                             07 1F 0003C          BLSSU   2$                                    ;
        55          60          10           00 ED 0003E          CMPZV   #0, #16, (REC_PTR), NT_ENDS           ; 3228
                                             19 1F 00043          BLS     3$                                    ;
                        51      FB37         CF 9E 00045 2$:      MOVAB   PAT$GET_NXT_LVT, R1                   ; 3243
                        51        08         AC D1 0004A          CMPL    ACCESS_FUNCTION, R1                   ;                                   ; F
                                             DD 12 0004E          BNEQ    1$                                    ;
        54          60          10           00 ED 00050          CMPZV   #0, #16, (REC_PTR), GL_NTS_BEGIN      ; 3245
                                             D6 1F 00055          BLSSU   1$                                    ;
        53          60          10           00 ED 00057          CMPZV   #0, #16, (REC_PTR), GL_NTS_END        ; 3246
                                             CF 1E 0005C          BGEQU   1$                                    ;
                                             52 D6 0005E 3$:      INCL    DELETE_COUNT                          ; 3249
                                60           01 B0 00060          MOVW    #1, (REC_PTR)                         ; 3250
                                             C8 11 00063          BRB     1$                                    ; 3220
                                50           52 D0 00065 4$:      MOVL    DELETE_COUNT, R0                      ; 3258
                                             04 00068             RET                                           ; 3259

; Routine Size:  105 bytes,    Routine Base:  _PAT$CODE + 048F
```

```
; 1428          3260  1 END                                   . End of module
; 1429          3261  0 ELUDOM


                                                    .EXTRN  LIB$SIGNAL

;                               PSECT SUMMARY
;
;       Name                    Bytes                   Attributes
;
;  _PAT$CODE                      1272  NOVEC,NOWRT,  RD ,  EXE,NOSHR,  LCL,  REL,  CON,NOPIC,ALIGN(2)
;  _PAT$OWN                          8  NOVEC,  WRT,  RD ,NOEXE,NOSHR,  LCL,  REL,  CON,NOPIC,ALIGN(2)
;  _PAT$PLIT                        27  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         4         0      1000          00:01.7


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


;                                       COMMAND QUALIFIERS

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

; Size:          1272 code + 35 data bytes
; Run Time:        00:40.3
; Elapsed Time:    02:07.0
; Lines/CPU Min:    4858
; Lexemes/CPU-Min: 28298
; Memory Used:  214 pages
; Compilation Complete
```

PATREB
LIS

PATSCA
LIS

PATSIO
LIS

PATRST
LIS

PATSPA
LIS

PATSSV
LIS