```
PPPPPPPPPPPP        AAAAAAAAA     TTTTTTTTTTTTTTT     CCCCCCCCCCCC HHH         HHH
PPPPPPPPPPPP        AAAAAAAAA     TTTTTTTTTTTTTTT     CCCCCCCCCCCC HHH         HHH
PPPPPPPPPPPP        AAAAAAAAA     TTTTTTTTTTTTTTT     CCCCCCCCCCCC HHH         HHH
PPP         PPP  AAA         AAA        TTT        CCC             HHH         HHH
PPP         PPP  AAA         AAA        TTT        CCC             HHH         HHH
PPP         PPP  AAA         AAA        TTT        CCC             HHH         HHH
PPP         PPP  AAA         AAA        TTT        CCC             HHH         HHH
PPP         PPP  AAA         AAA        TTT        CCC             HHH         HHH
PPP         PPP  AAA         AAA        TTT        CCC             HHH         HHH
PPPPPPPPPPPP     AAA         AAA        TTT        CCC             HHHHHHHHHHHHHHH
PPPPPPPPPPPP     AAA         AAA        TTT        CCC             HHHHHHHHHHHHHHH
PPPPPPPPPPPP     AAA         AAA        TTT        CCC             HHHHHHHHHHHHHHH
PPP              AAAAAAAAAAAAAAA        TTT        CCC             HHH         HHH
PPP              AAAAAAAAAAAAAAA        TTT        CCC             HHH         HHH
PPP              AAAAAAAAAAAAAAA        TTT        CCC             HHH         HHH
PPP              AAA         AAA        TTT        CCC             HHH         HHH
PPP              AAA         AAA        TTT        CCC             HHH         HHH
PPP              AAA         AAA        TTT        CCC             HHH         HHH
PPP              AAA         AAA        TTT           CCCCCCCCCCCC HHH         HHH
PPP              AAA         AAA        TTT           CCCCCCCCCCCC HHH         HHH
PPP              AAA         AAA        TTT           CCCCCCCCCCCC HHH         HHH
```

FILEID**PATSSV

K 15

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

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
```

PA
VO

;

```
  0 MODULE PATSSV (
L 0               %IF %VARIANT EQL 1
  0               %THEN
  0                       ADDRESSING_MODE (EXTERNAL = LONG_RELATIVE, NONEXTERNAL = LONG_RELATIVE),
  0               %FI
  0               IDENT = 'V04-000') =
  1 BEGIN
  1
  1 !****************************************************************************
  1 !*                                                                          *
  1 !*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
  1 !*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
  1 !*  ALL RIGHTS RESERVED.                                                    *
  1 !*                                                                          *
  1 !*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
  1 !*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
  1 !*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
  1 !*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
  1 !*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
  1 !*  TRANSFERRED.                                                            *
  1 !*                                                                          *
  1 !*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
  1 !*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
  1 !*  CORPORATION.                                                            *
  1 !*                                                                          *
  1 !*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
  1 !*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
  1 !*                                                                          *
  1 !*                                                                          *
  1 !****************************************************************************
  1 !
  1 ! FACILITY:     PATCH
  1 !
  1 !++
  1 ! FUNCTIONAL DESCRIPTION:
  1 !
  1 !       CALLS TO STARLET OPERATING SYSTEM I/O SERVICES.  INCLUDE
  1 !       ASSIGNMENT AND DEASSIGNMENT OF CHANNELS, READS, WRITES.
  1 !
  1 ! Version:      V02-018
  1 !
  1 ! History:
  1 !       Author:
  1 !               Carol Peters, 21 Sep 1976: Version 01
  1 !
  1 !       Modified by:
  1 !               Kevin Pammett, 15-dec-77: Version 25
  1 !
  1 !       V03-001 MTR0012         Mike Rhodes                     16-Aug-1982
  1 !               Modify file names to remove duplicate file name useage
  1 !               between code and require files.
  1 !
  1 !       V02-018 PCG0001         Peter George                    02-FEB-1981
  1 !               Add require statement for LIB$:PATDEF.REQ
  1 !
  1 !       V0117   CNH0016         Chris Hume                      4-Oct-1979      14:00
  1 !               Fix PAT$OUT_SYM_VAL to properly insert a leading zero where
```

```
1 !                     required to distinguish hex constants from symbols.  This cures
1 !                     several anomalies.  (PATMAI.B32 02.26)
1 !
1 ! Revision History:
1 !
1 ! NO    DATE            PROGRAMMER                      PURPOSE
1 ! --    ----            ----------                      -------
1 !
1 ! 00    20-DEC-77       K.D. MORSE                      ADAPT VERSION 25 FOR PATCH.
1 !  01   27-DEC-77       K.D. MORSE                      CHANGE PAT$OUT_TYP_VAL CALLS TO
1 !                                                       FOR$CVT ... TO CALL PAT$GET_VALUE.
1 !  02   2-JAN-78        K.D. MORSE                      ADD CHECKS FOR IMAGE WITH NO SYMBOLS.
1 !  03   4-JAN-78        K.D. MORSE                      FOR OUTPUTTING COMPLEX NUMBERS,
1 !                                                       SURROUND THEM IN PARENTHESIS, AND
1 !                                                       REMOVE THE SPURIOUS QUOTES
1 !                                                       AROUND THE COMMA. (26)
1 !                                                       NO CHANGES FOR 27.
1 !  04   28-FEB-78       K.D. MORSE                      NO CHANGES FOR 28-29.
1 !  05   02-MAR-78       K.D. MORSE                      NO CHANGES FOR 30-31.
1 !  06   17-MAR-78       K.D. MORSE                      ADD CODE TO OUTPUT A LEADING 0
1 !                                                       FOR NEGATIVE HEX NUMBERS TO
1 !                                                       PAT$OUT_NUM_VAL.  THE CODE WAS
1 !                                                       TO DO THIS IN PAT$INS_DECODE
1 !                                                       WAS INCORRECT AND THERFORE REMOVED.
1 !  07   24-MAR-78       K.D. MORSE                      NO CHANGES FOR VERS 32-34.
1 !  08   30-MAR-78       K.D. MORSE                      ADD "PC-RELATIVE" ARRAYS.
1 !  09   07-APR-78       K.D. MORSE                      ADD PAT$PV_TO_CS, AND CHANGE
1 !                                                       PAT$PRINT_PATH TO USE IT (35).
1 !                                                       OUT_SYM_VAL NOW CONSIDERS
1 !                                                       DEFINE SYMBOLS (FOR EXACT
1 !                                                       MATCHES ONLY) (36).
1 !                                                       NO CHANGES FOR 37.
1 !  10   25-APR-78       K.D. MORSE                      CONVERT TO NATIVE COMPILER.
1 !  11   17-MAY-78       K.D. MORSE                      SHOW_MODU NOW SKIPS THE GLOBAL
1 !                                                       MC. (38)
1 !                                                       SHOW_MODU PRINTS ERROR MESSAGE
1 !                                                       IF THERE IS NO DST. (39)
1 !  12   18-MAY-78       K.D. MORSE                      NO CHANGES FOR VERS 40-41.
1 !  13   13-JUN-78       K.D. MORSE                      ADD FAO COUNTS TO SIGNALS.
1 !  14   27-JUN-78       K.D. MORSE                      ADD LEADING ZEROS FOR HEX
1 !                                                       OUTPUT IN OUT_SYM_VAL. (42)
1 !  15   28-JUN-78       K.D. MORSE                      NO CHANGES FOR VERS 43-48.
1 !  16   29-JUN-78       K.D. MORSE                      NO CHANGES FOR VERS 49.
1 !                                                       SHOW MODULE NOW PRINTS
1 !                                                       ALL NUMBERS AS DECIMAL (50).
1 !                                                       NO CHANGES FOR VERS 51.
1 !
1 !--
```

```
1 !
1 ! TABLE OF CONTENTS:
1 !
1
1 FORWARD ROUTINE
1         PAT$PV_TO_CS : NOVALUE,                    ! Encode a pathvector into a buffer.
1         PAT$FAO_PUT : NOVALUE,                     ! Stuff output into current output buffer.
1         PAT$MODULE_SIZE,                           ! Estimate the RST size of a module.
1         PAT$OUT_NUM_VAL : NOVALUE,                 ! Output of numeric values.
1         PAT$OUT_PUT : NOVALUE,                     ! Actually write out a line to tty
1         PAT$OUT_SYM_VAL : NOVALUE,                 ! Output values either in numeric
1                                                    ! or in symbolic form.
1         PAT$SHOW_MODULE : NOVALUE,                 ! List off the module chain.
1         PAT$SHOW_SCOPE : NOVALUE,                  ! Print out the current CSP.
1         PAT$PRINT_PATH : NOVALUE;                  ! Print out a symbol pathname
1
1 LIBRARY 'SYS$LIBRARY:LIB.L32';
1 REQUIRE 'SRC$:VXSMAC.REQ';
1 REQUIRE 'LIB$:PATDEF.REQ';                         ! Defines literals
1 REQUIRE 'LIB$:PATMSG.REQ';
1 REQUIRE 'SRC$:PATPCT.REQ';
1 REQUIRE 'SRC$:PATGEN.REQ';
1 REQUIRE 'SRC$:SYSLIT.REQ';
1 REQUIRE 'SRC$:SYSSER.REQ';
```

```
SWITCHES LIST (SOURCE);

EXTERNAL ROUTINE
        PAT$fao_out;            ! formats a line and outputs to the terminal

```

```
1 REQUIRE 'SRC$:PATRTS.REQ';
1 REQUIRE 'SRC$:BSTRUC.REQ';
1 REQUIRE 'SRC$:DLLNAM.REQ';
1 REQUIRE 'SRC$:PREFIX.REQ';
1 REQUIRE 'SRC$:PATPRE.REQ';
1
1 EXTERNAL ROUTINE
1         PAT$FIND_VAL,                                     ! VAL_TO_SYM on define symbols
1         PAT$GET_BOUNDS : NOVALUE,                         ! Understand array descriptors
1         PAT$GET_VALUE : NOVALUE,                          ! Takes unmapped address and returns content
1         PAT$ADD_NT_T_PV,                                  ! Built pathname vectors
1         PAT$REPORT_FREE,                                  ! Bytes remaining in free storage
1         PAT$VAL_TO_SYM,                                   ! Translate values to their
1                                                           ! symbolic equivalent
1         PAT$SYMBOL_VALU,
1         SYS$FAOL: ADDRESSING_MODE (ABSOLUTE);             ! System service to do formatted output
1
1 EXTERNAL
1         PAT$GL_SYMTBPTR,                                  ! Pointer to default symbol table
1         PAT$GL_OLDLABLS,                                  ! Pointer to old labels symbol table
1         PAT$GL_SYMHEAD,                                   ! Pointer to user-defined symbol table
1         PAT$GB_SYMBOLS,                                   ! Indicator if image has symbols
1         PAT$CP_OUT_STR: REF VECTOR[, BYTE],               ! Points into current output buffer
1         PAT$GB_MOD_PTR: REF VECTOR[, BYTE],               ! Points to current I/O mode settings
1         PAT$GL_BUF_SIZ,                                   ! Holds current character count in output bu
1         PAT$GL_CSP_PTR  : ref pathname_vector,            ! The current scope position (CSP)
1         PAT$GL_LAST_VAL,                                  ! The last value displayed
1         PAT$GL_MC_PTR   : REF MC_RECORD,                  ! Pointer to the module chain (MC).
1         PAT$GL_NEXT_LOC;                                  ! Next location to display
```

```
1 GLOBAL ROUTINE PAT$FAO_PUT( STRING, ARGUMENTS ) : NOVALUE =
1
1 !++
1 ! FUNCTIONAL DESCRIPTION:
1 !
1 !       Do just what $FAO does, only here we work in co-operation with a
1 !       global character buffer into which we are encoding arbitrary lines
1 !       of output.
1 !
1 !       All console output done within PATCH should use this routine to build
1 !       output lines.  The only other I/O call which PATCH should be making
1 !       (for the console) is to PAT$OUT_PUT, which simply says "put out whatever
1 !       PAT$FAL_PUT built".
1 !
1 ! CALLING SEQUENCE:
1 !
1 !       PAT$FAO_PUT ();
1 !
1 ! INPUTS:
1 !
1 !       STRING    - A counted string which contains the directives for $FAO.
1 !       ARGUMENTS - The arguments for $FAO.
1 !
1 ! IMPLICIT INPUTS:
1 !
1 !       PAT$CP_OUT_STR  - Pointer to where we are in the current output buffer.
1 !       PAT$GL_BUF_SIZ  - Count of characters in output buffer.
1 !
1 ! OUTPUTS:
1 !
1 !       The $FAO output is put into the output buffer.
1 !
1 ! IMPLICIT OUTPUTS:
1 !
1 !       The global character pointer is incremented so that it points
1 !       (as always) to the next available place in the output buffer.
1 !       The buffer count variable is incremented by the size of this string.
1 !
1 ! ROUTINE VALUE:
1 !
1 !       none
1 !
1 ! SIDE EFFECTS:
1 !
1 !       none
1 !--
1
2 BEGIN
2 MAP
2       STRING : REF VECTOR[,BYTE];
2 LOCAL
2       INP_DESC : VECTOR[2],                           ! Input desc for $FAO
2       OUT_DESC : VECTOR[2],                           ! Output desc for $FAO
2       STR_SIZE : WORD;                                ! $FAO returns output size here
2
2 !++
2 ! Build the descriptors that $FAO wants, ask it to do the encoding,
```

```
;   218          2472  2 ! copying the output into our global output buffer, and finally update the
;   219          2473  2 ! global pointer to the next free character position in the buffer.
;   220          2474  2 !-
;   221          2475  2 INP_DESC[0] = .STRING[0];
;   222          2476  2 INP_DESC[1] = STRING[1];
;   223          2477  2 OUT_DESC [0] = TTY_OUT_WIDTH - 1 - .PAT$GL_BUF_SIZ;
;   224          2478  2 OUT_DESC[1] = .PAT$CP_OUT_STR;
;   225          2479  2 SYS$FAOL( INP_DESC, STR_SIZE, OUT_DESC, ARGUMENTS );
;   226          2480  2 PAT$CP_OUT_STR = .PAT$CP_OUT_STR + .STR_SIZE;
;   227          2481  2 PAT$GL_BUF_SIZ = .PAT$GL_BUF_SIZ + .STR_SIZE;
;   228          2482  1 END;

                                                        .TITLE  PATSSV
                                                        .IDENT  \V04-000\

                                                  ISE$C_SIZE==      20
                                                  TXT$C_SIZE==      4
                                                  PAL$C_SIZE==      16
                                                  ASD$C_SIZE==      9
                                                  FWR$C_SIZE==      24
                                                        .EXTRN  PAT$FAO_OUT, PAT$FIND_VAL
                                                        .EXTRN  PAT$GET_BOUNDS, PAT$GET_VALUE
                                                        .EXTRN  PAT$ADD_NT_T_PV
                                                        .EXTRN  PAT$REPORT_FREE
                                                        .EXTRN  PAT$VAL_TO_SYM, PAT$SYMBOL_VALU
                                                        .EXTRN  SYS$FAOL, PAT$GL_SYMTBPTR
                                                        .EXTRN  PAT$GL_OLDLABLS
                                                        .EXTRN  PAT$GL_SYMHEAD, PAT$GB_SYMBOLS
                                                        .EXTRN  PAT$CP_OUT_STR, PAT$GB_MOD_PTR
                                                        .EXTRN  PAT$GL_BUF_SIZ, PAT$GL_CSP_PTR
                                                        .EXTRN  PAT$GL_LAST_VAL
                                                        .EXTRN  PAT$GL_MC_PTR, PAT$GL_NEXT_LOC
                                                        .WEAK   ACCESS_CHECK

                                                        .PSECT  _PAT$CODE,NOWRT,2

                                   000C 00000           .ENTRY  PAT$FAO_PUT, Save R2,R3                      ; 2415
                   53 00000000G  EF  9E 00002           MOVAB   PAT$GL_BUF_SIZ, R3                           ;
                   52 00000000G  EF  9E 00009           MOVAB   PAT$CP_OUT_STR, R2                           ;
                             5E  14  C2 00010           SUBL2   #20, SP                                      ;
                   0C        AE  04  BC  9A 00013       MOVZBL  @STRING, INP_DESC                            ; 2475
         10    AE  04        AC      01  C1 00018       ADDL3   #1, STRING, INP_DESC+4                       ; 2476
         04    AE 00000083   8F      63  C3 0001E       SUBL3   PAT$GL_BUF_SIZ, #131, OUT_DESC               ; 2477
                   08        AE      62  D0 00027       MOVL    PAT$CP_OUT_STR, OUT_DESC+4                   ; 2478
                             08  AC  9F 0002B           PUSHAB  ARGUMENTS                                    ; 2479
                             08  AE  9F 0002E           PUSHAB  OUT_DESC                                     ;
                             08  AE  9F 00031           PUSHAB  STR_SIZE                                     ;
                             18  AE  9F 00034           PUSHAB  INP_DESC                                     ;
               00000000G  9F     04  FB 00037           CALLS   #4, @#SYS$FAOL                               ;
                          50     6E  3C 0003E           MOVZWL  STR_SIZE, R0                                 ; 2480
                          62     50  C0 00041           ADDL2   R0, PAT$CP_OUT_STR                           ;
                          50     6E  3C 00044           MOVZWL  STR_SIZE, R0                                 ; 2481
                          63     50  C0 00047           ADDL2   R0, PAT$GL_BUF_SIZ                           ;
                                 04 0004A               RET                                                  ; 2482

; Routine Size:  75 bytes,    Routine Base:  _PAT$CODE + 0000
```

```
  1 GLOBAL ROUTINE PAT$MODULE_SIZE( MC_PTR ) =
  1
  1 !++
  1 ! Functional Description:
  1 !
  1 !       Given an MC pointer to a given module, estimate how many bytes of RST
  1 !       storage would be required to SET (initialize) this module into the RST.
  1 !
  1 ! Calling Sequence:
  1 !
  1 !       The routine is simply called to pick up the returned value.  The reason
  1 !       why it is global is because it is called in PAT$SHOW_MODULE, of
  1 !       PATSSV.B32, and in ADD_MODULE, of PATBLD.B32.
  1 !
  1 ! Formal Parameters:
  1 !
  1 !       MC_PTR  -An MC pointer which tells us all we need
  1 !                to know about the indicated module.
  1 !
  1 ! Implicit Inputs:
  1 !
  1 !       The RST storage we consider is independent of GLOBAL
  1 !       considerations.  This means that in all likelihood we overestimate.
  1 !       (We guarantee NOT to underestimate in any case.)
  1 !
  1 !       How the storage manager works, and how we
  1 !       use it to allocate all RST storage.
  1 !
  1 ! Implicit Outputs:
  1 !
  1 !       none.
  1 !
  1 ! Return Value:
  1 !
  1 !       The number of bytes required to add the module
  1 !       to the Runtime Symbol Table data structures.
  1 !
  1 !--
  1
  2 BEGIN
  2
  2 MAP
  2         MC_PTR : REF MC_RECORD;
  2
  2 LOCAL
  2         TOTAL_BYTES;
  2
  2 !++
  2 ! Macro to calculate the total number of bytes
  2 ! taken for so-called vector storage (which
  2 ! is how we allocate NTs, LVTs, and SATs).
  2 !--
  2 MACRO
  2
M 2         VEC_STORAGE_FOR( RECORDS, SIZE ) = (%UPVAL +              ! The storage manager's overhead.
  2                 ( (RST_UNITS( RECORDS * SIZE )) * %UPVAL ))%; ! Rounded up vector st
  2
```

```
;  287        2540  2 !++
;  288        2541  2 ! Tally up the total number of bytes needed to add the NTs, LVTs, and
;  289        2542  2 ! SATs for the indicated module.  First initialize counter.
;  290        2543  2 !--
;  291        2544  2 TOTAL_BYTES = 0;
;  292        2545  2
;  293        2546  2 !++
;  294        2547  2 ! Although it is not true, we must assume that NT entries are fixed-size.
;  295        2548  2 !--
;  296        2549  2 TOTAL_BYTES = VEC_STORAGE_FOR( .MC_PTR [MC_NAMES], RST_NT_SIZE );
;  297        2550  2
;  298        2551  2 !++
;  299        2552  2 ! Add on that taken for SATs and LVTs.
;  300        2553  2 !--
;  301        2554  2 TOTAL_BYTES = .TOTAL_BYTES
;  302        2555  3               + VEC_STORAGE_FOR( .MC_PTR [MC_STATICS], RST_SAT_SIZE )
;  303        2556  2               + VEC_STORAGE_FOR( .MC_PTR [MC_LITERALS], RST_LVT_SIZE );
;  304        2557  2
;  305        2558  2 !++
;  306        2559  2 ! Return the size.
;  307        2560  2 !--
;  308        2561  2 RETURN(.TOTAL_BYTES);
;  309        2562  1 END;


                                                   .EXTRN   PAT$GL_RST_BEGN

                                 000C 00000        .ENTRY   PAT$MODULE_SIZE, Save R2,R3          ; 2483
                               50   D4 00002        CLRL     TOTAL_BYTES                          ; 2544
        52    04  AC 00000000G EF   C1 00004        ADDL3    PAT$GL_RST_BEGN, MC_PTR, R2          ; 2549
        51    08  A2           2C   C5 0000D        MULL3    #44, 8(R2), R1                       ;
                  51           03   C0 00012        ADDL2    #3, R1                               ;
                  51           04   C6 00015        DIVL2    #4, R1                               ;
        50        51           02   78 00018        ASHL     #2, R1, TOTAL_BYTES                  ;
                  50           04   C0 0001C        ADDL2    #4, TOTAL_BYTES                      ;
        51    31  A2           0A   C5 0001F        MULL3    #10, 49(R2), R1                      ; 2555
                  51           03   C0 00024        ADDL2    #3, R1                               ;
                  51           04   C6 00027        DIVL2    #4, R1                               ;
                  53         6041   DE 0002A        MOVAL    (TOTAL_BYTES)[R1], R3                ; 2556
        52    35  A2           06   C5 0002E        MULL3    #6, 53(R2), R2                       ;
                  51       03  A2   9E 00033        MOVAB    3(R2), R1                            ;
                  51           04   C6 00037        DIVL2    #4, R1                               ;
                  50     08 A341    DE 0003A        MOVAL    8(R3)[R1], TOTAL_BYTES               ;
                               04   0003F           RET                                           ; 2562

; Routine Size:  64 bytes,    Routine Base:  _PAT$CODE + 004B
```

```
GLOBAL ROUTINE PAT$OUT_NUM_VAL ( VALUE, NEW_LENGTH, NEW_RADIX, ZERO_FILL) : NOVALUE =

!++
! Functional Description:
!
!       Format the given value according to the current output mode settings.
!       The formatted value is inserted into the output buffer described by
!       PAT$CP_OUT_STR and PAT$GL_BUF_SIZ.  If the value is formatted in
!       hexadecimal and negative and ZERO_FILL is TRUE, then a leading zero
!       is output.
!
! Calling Sequence:
!
!       PAT$OUT_NUM_VAL (VALUE, NEW_LENGTH, NEW_RADIX, ZERO_FILL)
!
! Inputs:
!
!       VALUE - The actual value we are to write out.
!       NEW_LENGTH - Either 0, or the MODE_LENGTH we should use.
!                    (0 => use current LENGTH, non-zero allows
!                     it to be overridden.)
!                    (The literal, NO_OVERRIDE = 0, is used for this.)
!       NEW_RADIX - Either 0, or the MODE_RADIX we should use.
!       ZERO_FILL - Indicator if leading zero should be output
!                         (TRUE = yes, FALSE = no)
!
! Implicit Inputs:
!
!       PAT$GB_MOD_PTR - points to the current mode data structure.
!       PAT$CP_OUT_STR - points into the output buffer at the first
!                        place in the buffer which is available.
!       PAT$GL_BUF_SIZ - holds the current number of characters in
!                        the buffer.
!
! Outputs:
!
!       The (numeric) character representation
!       of the value is encoded into the output buffer.
!
! Implicit Outputs:
!
!       The buffer pointer, PAT$CP_OUT_STR, is incremented.
!       The buffer count, PAT$GL_BUF_SIZ, is incremented.
!
! Routine Value:
!
!       NOVALUE.
!
! Side Effects.
!
!       Negative hex numbers may be preceded by a leading 0.
!--

BEGIN

LOCAL
        USE_LENGTH,                                             ! The MODE_LENGTH we actually use
```

```
         USE_RADIX;                                             ! The MODE_RADIX we actually use

OWN
         FORMAT : VECTOR[4,BYTE]                                ! Build format string here
                         INITIAL( BYTE( %ASCIC '!??' ) );

!++
! Assume that we are passed an override for the LENGTH and RADIX,
! but revert to the globally-set one if this is untrue.
!--
IF ((USE_LENGTH = .NEW_LENGTH) EQL 0)
THEN
         USE_LENGTH = .PAT$GB_MOD_PTR  [MODE_LENGTH];
IF ((USE_RADIX = .NEW_RADIX) EQL 0)
THEN
         USE_RADIX = .PAT$GB_MOD_PTR  [MODE_RADIX];

!++
! Now just build the required 2-character format string descriptor,
! based upon the current setting of the mode length and radix.
!
! We assume signed, for decimal output, and take longword hexadecimal either
! when that is explicitly the case, or when some unknown length or radix is indicated.
!--
FORMAT[2] = ( SELECTONE .USE_RADIX OF
                 SET
                 [OCTAL_RADIX]:          'O';
                 [DECIMAL_RADIX]:        'S';
                 [OTHERWISE]:            'X';
                 TES
             );

FORMAT[3] = ( SELECTONE .USE_LENGTH OF
                 SET
                 [BYTE_LENGTH]:  'B';
                 [WORD_LENGTH]:  'W';
                 [OTHERWISE]:    'L';
                 TES
             );

!++
! Now check if the output radix is hexadecimal and if a leading zero is needed.
! This is required to make PATCH output acceptable as input.  Hexadecimal
! numbers require the leading character to be a numeric.
!--
IF (.FORMAT[2] EQLU 'X') AND (.ZERO_FILL) AND
   (.(VALUE)<0, .USE_LENGTH*8, 1> GTRA %X'9FFFFFFF')
THEN
         BEGIN
         PAT$CP_OUT_STR[0] = '0';
         PAT$CP_OUT_STR = .PAT$CP_OUT_STR + 1;
         PAT$GL_BUF_SIZ = .PAT$GL_BUF_SIZ + 1;
         END;

PAT$FAO_PUT ( FORMAT, .VALUE );

END;
```

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

                            3F  3F  21  03  00000 FORMAT: .ASCII  <3>\!??\                                               ;

                                                          .PSECT  _PAT$CODE,NOWRT,2

                                       0004 00000         .ENTRY  PAT$OUT_NUM_VAL, Save R2                            ; 2563
                       52 00000000'  EF  9E 00002         MOVAB   FORMAT+2, R2                                        ;
                       50        08  AC  D0 00009         MOVL    NEW_LENGTH, USE_LENGTH                              ; 2630
                                 0B  12 0000D             BNEQ    1$                                                  ;
                       51 00000000G  EF  D0 0000F         MOVL    PAT$GB_MOD_PTR, R1                                  ; 2632
                       50        01  A1  9A 00016         MOVZBL  1(R1), USE_LENGTH                                   ;
                       51        0C  AC  D0 0001A 1$:     MOVL    NEW_RADIX, USE_RADIX                                ; 2633
                                 07  12 0001E             BNEQ    2$                                                  ;
                       51 00000000G  FF  9A 00020         MOVZBL  @PAT$GB_MOD_PTR, USE_RADIX                          ; 2635
                       08            51  D1 00027 2$:     CMPL    USE_RADIX, #8                                       ; 2646
                                 06  12 0002A             BNEQ    3$                                                  ;
                       51        4F  8F  9A 0002C         MOVZBL  #79, R1                                             ;
                                 0F  11 00030             BRB     5$                                                  ;
                       0A            51  D1 00032 3$:     CMPL    USE_RADIX, #10                                      ; 2647
                                 06  12 00035             BNEQ    4$                                                  ;
                       51        53  8F  9A 00037         MOVZBL  #83, R1                                             ;
                                 04  11 0003B             BRB     5$                                                  ;
                       51        58  8F  9A 0003D 4$:     MOVZBL  #88, R1                                             ; 2648
                       62            51  90 00041 5$:     MOVB    R1, FORMAT+2                                        ; 2644
                       01            50  D1 00044         CMPL    USE_LENGTH, #1                                      ; 2654
                                 06  12 00047             BNEQ    6$                                                  ;
                       51        42  8F  9A 00049         MOVZBL  #66, R1                                             ;
                                 0F  11 0004D             BRB     8$                                                  ;
                       02            50  D1 0004F 6$:     CMPL    USE_LENGTH, #2                                      ; 2655
                                 06  12 00052             BNEQ    7$                                                  ;
                       51        57  8F  9A 00054         MOVZBL  #87, R1                                             ;
                                 04  11 00058             BRB     8$                                                  ;
                       51        4C  8F  9A 0005A 7$:     MOVZBL  #76, R1                                             ; 2656
                 01    A2            51  90 0005E 8$:     MOVB    R1, FORMAT+3                                        ; 2652
                 58    8F            62  91 00062         CMPB    FORMAT+2, #88                                       ; 2665
                                 29  12 00066             BNEQ    9$                                                  ;
                       25        10  AC  E9 00068         BLBC    ZERO_FILL, 9$                                       ;
                       50            08  C4 0006C         MULL2   #8, R0                                              ; 2666
   51       04    AC   50            00  EE 0006F         EXTV    #0, R0, VALUE, R1                                   ;
           9FFFFFFF    8F            51  D1 00075         CMPL    R1, #-1610612737                                    ;
                                 13  1B 0007C             BLEQU   9$                                                  ;
           00000000G   FF            30  90 0007E         MOVB    #48, @PAT$CP_OUT_STR                                ; 2669
                         00000000G   EF  D6 00085         INCL    PAT$CP_OUT_STR                                      ; 2670
                         00000000G   EF  D6 0008B         INCL    PAT$GL_BUF_SIZ                                      ; 2671
                                 04  AC  DD 00091 9$:     PUSHL   VALUE                                               ; 2674
                                 FE  A2  9F 00094         PUSHAB  FORMAT                                              ;
                FED9   CF            02  FB 00097         CALLS   #2, PAT$FAO_PUT                                     ;
                                     04 0009C             RET                                                         ; 2676

; Routine Size:  157 bytes,    Routine Base:  _PAT$CODE + 008B
```

```
1 GLOBAL ROUTINE PAT$OUT_PUT ( BUFFER ) : NOVALUE =
1
1 !++
1 ! Functional Description:
1 !
1 !       Cause the current output buffer to be actually
1 !       output to the console.
1 !
1 ! Calling Sequence:
1 !
1 !       PAT$OUT_PUT ()
1 !
1 ! Inputs:
1 !
1 !       BUFFER - Pointer to the beginning of the current output
1 !                buffer.  this is supposed to be a counted
1 !                string except that noone has supplied the count yet.
1 !                ie we expect that the actual string to be printed
1 !                starts in byte BUFFER+1.
1 !
1 ! Implicit Inputs:
1 !
1 !       PAT$CP_OUT_STR - Points into this output buffer at the first
1 !                        place in the buffer which is NOT to be printed.
1 !       PAT$GL_BUF_SIZ - Holds count of characters in buffer
1 !
1 ! Outputs:
1 !
1 !       The string is printed, exactly as it is in the
1 !       buffer - ie, it should contain whatever carriage control
1 !       you want.
1 !
1 ! Implicit Outputs:
1 !
1 !       The buffer pointer, PAT$CP_OUT_STR, is set to zero.  This will
1 !       help routines know if they are to use their own buffer or
1 !       if one is already partially built.
1 ! ***   I don't think we have ever actually used the above
1 !       'feature', and feel that this routine ought, instead, set
1 !       up for more PAT$FAO_PUT calls on the current buffer when we
1 !       are called to output it.
1 !
1 ! Routine Value:
1 !
1 !       NOVALUE.
1 !
1 ! Side Effects:
1 !
1 !       none.
1 !--
1
2 BEGIN
2
2 MAP
2       BUFFER : REF VECTOR[,BYTE];                              ! Points to output buffer
2
2 !++
```

```
;  483          2734  2 ! Fill in the count, and pass it to QIO.
;  484          2735  2 !--
;  485          2736  2 BUFFER [0] = .PAT$GL_BUF_SIZ;
;  486          2737  2 $FAO_TT_OUT ( '!AC', .BUFFER );
;  487          2738  2 PAT$CP_OUT_STR = 0;
;  488          2739  1 END;


                                                            .PSECT  _PAT$PLIT,NOWRT,NOEXE,0

                                        03  00000 P.AAA:    .BYTE   3                                    ;
                                43  41  21  00001           .ASCII  \!AC\                                ;


                                                            .PSECT  _PAT$CODE,NOWRT,2

                                      0000  00000           .ENTRY  PAT$OUT_PUT, Save nothing            ; 2677
                   04     BC 00000000G  EF  90  00002       MOVB    PAT$GL_BUF_SIZ, @BUFFER              ; 2736
                                    04  AC  DD  0000A       PUSHL   BUFFER                               ; 2737
                            00000000'   EF  9F  0000D       PUSHAB  P.AAA                                ;
           00000000G  EF                02  FB  00013       CALLS   #2, PAT$FAO_OUT                      ;
                            00000000G   EF  D4  0001A       CLRL    PAT$CP_OUT_STR                       ; 2738
                                            04  00020       RET                                          ; 2739

; Routine Size:  33 bytes,    Routine Base:  _PAT$CODE + 0128
```

```
GLOBAL ROUTINE PAT$OUT_SYM_VAL ( VALUE, NEW_LENGTH, NEW_RADIX ) : NOVALUE =

!++
! Functional Description:
!
!       Write out the given value according to the current
!       mode settings - including the current mode setting
!       for the [NO]SYMBOLS flag.  (By 'write out', we mean
!       'encode into the output buffer').
!
! Calling Sequence:
!
!       PAT$OUT_SYM_VAL ()
!
! Inputs:
!
!       VALUE - The actual value we are to write out.
!       NEW_LENGTH - Either 0, or the MODE_LENGTH we should use.
!                   (0 => use current LENGTH, non-zero allows
!                    it to be overridden.)
!                   (The literal, NO_OVERRIDE = 0, is used for this.)
!       NEW_RADIX - Either 0, or the MODE_RADIX we should use.
!
! Implicit Inputs:
!
!       PAT$GB_MOD_PTR - Points to the current mode data structure.
!       PAT$CP_OUT_STR - Points into the output buffer at the first
!                        place in the buffer which is available.
!       PAT$GL_BUF_SIZ - Holds the current number of characters in
!                        the buffer.
!
! Outputs:
!
!       The symbolic or numeric (or symbolic+residue) representation
!       of the value is encoded into the output buffer.
!
! Implicit Outputs:
!
!       The buffer pointer, PAT$CP_OUT_STR, is incremented.
!       The buffer count, PAT$GL_BUF_SIZ, is incremented.
!
! Routine Value:
!
!       NOVALUE.
!
! Side Effects.
!       The VALUE is printed out.
!--

BEGIN

LOCAL
                ARRAY_DESC_ADDR,                        ! The address of an array descriptor,
                                                        ! if we symbolize an array reference.
                NT_PTR : REF NT_RECORD,                 ! A pointer to the name table (NT) entry
                                                        ! which corresponds to the symbol which we
                                                        ! deal with if we do symbolic output.
```

```
2                       REAL_VALUE,                             ! Used to recover the actual or real value
2                                                               ! of the symbol we may base our symbolic
2                                                               ! output off.
2                       PATH_VEC : PATHNAME_VECTOR;             ! Symbol path vector used to contain the
2                                                               ! whole pathname assocaited with the
2                                                               ! symbol pointed to by NT_PTR, above.
2
2 !++
2 ! First, see if the value is an exact match to a DEFine type symbol.  Note that
2 ! these symbols have nothing to do with the so-called RST and so much be handled
2 ! specially.  First look in the current label symbol table, then in the old
2 ! label table, and lastly in the user-defined symbol table.
2 !--
3 IF (.PAT$GB_MOD_PTR[MODE_SYMBOLS])
2 THEN
3       BEGIN
3       LOCAL
3               TEMP_SYMTB,
3               INDEX;
3
3       INDEX = PAT$FIND_VAL(.VALUE,TRUE);
4       IF (.INDEX EQL 0)
3       THEN
4               BEGIN
4               TEMP_SYMTB = .PAT$GL_SYMTBPTR;                  ! Remember current label table
4               PAT$GL_SYMTBPTR = .PAT$GL_OLDLABLS;             ! Use old labels symbol table
4               INDEX = PAT$FIND_VAL(.VALUE,TRUE);
5               IF (.INDEX EQL 0)
4               THEN
5                       BEGIN
5                       PAT$GL_SYMTBPTR = .PAT$GL_SYMHEAD;      ! Use user-defined symbol table
5                       INDEX = PAT$FIND_VAL(.VALUE,TRUE);
4                       END;
4               PAT$GL_SYMTBPTR = .TEMP_SYMTB;                  ! Restor current label table
3               END;
4       IF (.INDEX NEQ 0)
3       THEN
4               BEGIN
4               !++
4               ! Found an exact match.  Print this and return an OK status.
4               !--
4               PAT$FAO_PUT(UPLIT(%ASCIC '!AD'),.SYM_CHCOUNT(.INDEX),SYM_NAME(.INDEX));
4               RETURN;
3               END;
2       END;
2 !++
2 ! Even though PAT$VAL_TO_SYM 'initializes' NT_PTR when we pass on its address,
2 ! we must still initialize it to zero because otherwise the top half of the
2 ! longword never gets cleared.  (PAT$VAL_TO_SYM thinks, and rightly so,
2 ! that NT_PTRs are only 2 bytes long, the problem is that BLISS doesn't realize
2 ! this when it passes .NT_PTR on as an actual parameter later on.)
2 !--
2 NT_PTR = 0;
2 ARRAY_DESC_ADDR = 0;
2
2 !++
2 ! First, see if we should even attempt symbolic output.
```

```
!--
IF (.PAT$GB_MOD_PTR [MODE_SYMBOLS]) AND (.PAT$GB_SYMBOLS)
THEN
        !++
        ! Next, ask VAL_TO_SYM for the NT-pointer which
        ! corresponds to the symbol we should use.
        ! The 'FALSE' says that we don't want LVTs
        ! (literals) to be considered.
        !--
        IF (PAT$VAL_TO_SYM( .VALUE, NT_PTR, FALSE))
        THEN
                !++
                ! Then, to be able and output ''symbol+offset'',
                ! we must be able to recover the actual value
                ! of the indicated symbol.
                !--
                IF (PAT$SYMBOL_VALU( .NT_PTR, REAL_VALUE))
                THEN
                        BEGIN
                        !++
                        ! See if the value picked up by PAT$SYMBOL_VALU is
                        ! really the address which is bound to the
                        ! symbol, or if it is the address of a descriptor.
                        ! Note that DST_PTR is invalid for NTs which
                        ! were created only for global symbols.
                        !--
                        IF (.NT_PTR [NT_UP_SCOPE] NEQ 0)
                        THEN
                                BEGIN
                                LOCAL
                                        BOUNDS : ARRAY_BNDS_DESC,
                                        DST_RECRD : REF DST_RECORD;

                                !++
                                ! Pick up the DST pointer so that we can see
                                ! if access is via descriptor.
                                !--
                                DST_RECRD = .NT_PTR [NT_DST_PTR];
                                IF (.DST_RECRD [DSTR_ACCES_TYPE] EQL ACCS_DESCRIPTOR) AND
                                   ( ( (.DST_RECRD[DSTR_ACCES_BASD] EQL ACCS_BASDIR) AND
                                       (.DST_RECRD[DSTR_ACCES_BREG] EQL 15) ) OR
                                     ( (.DST_RECRD[DSTR_ACCES_BASD] EQL 0) AND
                                       (.DST_RECRD[DSTR_ACCES_BREG] EQL 0) ) )
                                THEN
                                        BEGIN
                                        !++
                                        ! REAL_VALUE, returned above, is actually
                                        ! the address of a descriptor. Modify this
                                        ! value to be the beginning virtual address
                                        ! of the array.
                                        !--
                                        ARRAY_DESC_ADDR = .REAL_VALUE;
                                        PAT$GET_BOUNDS( .ARRAY_DESC_ADDR, BOUNDS);
                                        REAL_VALUE = .BOUNDS [ARRAY_ADDRESS];
                                        END;
                                END;
```

```
	!++
	! For unbounded symbols, we don't
	! allow "symbol+offset" unless "offset"
	! is less than a maximum value.
	!--
	IF (((.VALUE - .REAL_VALUE) LSSA RST_MAX_OFFSET)
	   OR (.NT_PTR [NT_IS_BOUNDED]))
	THEN
		BEGIN
		!++
		! We have all we need now, so just
		! encode the characters into the
		! output stream. To do this we
		! must first build a pathname vector
		! to correspond to the NT scope chain we
		! implicitly now have.
		!--
		PAT$ADD_NT_T_PV( .NT_PTR, PATH_VEC );

		!++
		! Then we simply print this vector.
		!--
		PAT$PRINT_PATH( PATH_VEC );

		!++
		! See if there is any residue - i.e. see if
		! "symbol[(...)]" is sufficient, or if we must
		! output "symbol[(...)]+offset".
		!--
		IF (.VALUE EQL .REAL_VALUE)
		THEN
			RETURN;				! No residue -> we're done.

		!++
		! The residue (offset) should
		! be printed in numeric form, preceeded
		! by a '+'.
		!--
		PAT$FAO_PUT(UPLIT(%ASCIC '+'));
		VALUE = .VALUE-.REAL_VALUE;

		!++
		! We should be able to simply
		! drop out of this part of the
		! code and have this done by PAT$OUT_NUM_VAL.
		! Until FAO can print HEX without the leading
		! 0s, we must do the following...
		!--
		IF (.PAT$GB_MOD_PTR [MODE_RADIX] EQL HEX_RADIX)
		THEN
			BEGIN
			!++
			! Special cludge to ignore leading 0s.
			! But to then put exactly 1 leading 0
			! when the rest begins with A-F.
			!--
			LOCAL
```

```
5                                               CLUDGE_SIZE,      ! # of digits for FAO.
5                                               CLUDGE_VAL;       ! First non-zero digit.
5
5                                       !++
5                                       ! We must always print at least 1 digit.
5                                       !--
5                                       DECRU I FROM 7
5                                       DO
6                                               BEGIN
6                                               CLUDGE_SIZE = .I+1;
6                                               IF (CLUDGE_VAL=.VALUE<.I*4,4,0>) NEQ 0
6                                               THEN
6                                                       EXITLOOP;
5                                               END;
5
5                                       IF .CLUDGE_VAL GEQU %X'A'
5                                       THEN
5                                               PAT$FAO_PUT(UPLIT(%ASCIC'0'));
5                                       PAT$FAO_PUT( UPLIT( %ASCIC '!#XL' ),
5                                               .CLUDGE_SIZE,.VALUE);
5                                       RETURN;
4                                       END;
3                               END;
2                       END;
2
2 !++
2 ! We didn't try to do symbolic output, or we failed completely at it, or
2 ! it went OK but we still have a 'residue'  value to output.
2 !--
2 PAT$OUT_NUM_VAL( .VALUE, .NEW_LENGTH, .NEW_RADIX, TRUE);
1 END;


                                                    .PSECT  _PAT$PLIT,NOWRT,NOEXE,0

                        44  41  21  03  00004 P.AAB:  .ASCII  <3>\!AD\                                ;
                        00  00  2B  01  00008 P.AAC:  .ASCII  <1>\+\<0><0>                            ;
                        00  00  30  01  0000C P.AAD:  .ASCII  <1>\0\<0><0>                            ;
        00  00  00  4C  58  23  21  04  00010 P.AAE:  .ASCII  <4>\!#XL\<0><0><0>                      ;


                                                    .PSECT  _PAT$CODE,NOWRT,2

                                  03FC  00000         .ENTRY  PAT$OUT_SYM_VAL, Save R2,R3,R4,R5,R6,R7,R8,-;  2740
                                                              R9                                      ;
                 59 00000000G  EF  9E  00002         MOVAB   PAT$GL_RST_BEGN, R9                     ;
                 58 00000000G  EF  9E  00009         MOVAB   PAT$GB_MOD_PTR, R8                      ;
                 57 00000000G  EF  9E  00010         MOVAB   PAT$FIND_VAL, R7                        ;
                 56      FE9C  CF  9E  00017         MOVAB   PAT$FAO_PUT, R6                         ;
                 55 00000000'  EF  9E  0001C         MOVAB   P.AAB, R5                               ;
                 54 00000000G  EF  9E  00023         MOVAB   PAT$GL_SYMTBPTR, R4                     ;
                 5E            3C  C2  0002A         SUBL2   #60, SP                                 ;
                 50            68  D0  0002D         MOVL    PAT$GB_MOD_PTR, R0                      ;  2810
                 44        02  A0  E9  00030         BLBC    2(R0), 3$                               ;
                               01  DD  00034         PUSHL   #1                                      ;  2817
```

```
                                     04    AC  DD 00036        PUSHL   VALUE                                   ;
                          67               02  FB 00039        CALLS   #2, PAT$FIND_VAL                        ;
                                           50  D5 0003C        TSTL    INDEX                                   ; 2818
                                           28  12 0003E        BNEQ    2$                                      ;
                          52               64  D0 00040        MOVL    PAT$GL_SYMTBPTR, TEMP_SYMTB             ; 2821
                          64 00000000G     EF  D0 00043        MOVL    PAT$GL_OLDLABLS, PAT$GL_SYMTBPTR        ; 2822
                                           01  DD 0004A        PUSHL   #1                                      ; 2823
                                     04    AC  DD 0004C        PUSHL   VALUE                                   ;
                          67               02  FB 0004F        CALLS   #2, PAT$FIND_VAL                        ;
                                           50  D5 00052        TSTL    INDEX                                   ; 2824
                                           0F  12 00054        BNEQ    1$                                      ;
                          64 00000000G     EF  D0 00056        MOVL    PAT$GL_SYMHEAD, PAT$GL_SYMTBPTR         ; 2827
                                           01  DD 0005D        PUSHL   #1                                      ; 2828
                                     04    AC  DD 0005F        PUSHL   VALUE                                   ;
                          67               02  FB 00062        CALLS   #2, PAT$FIND_VAL                        ;
                          64               52  D0 00065 1$:    MOVL    TEMP_SYMTB, PAT$GL_SYMTBPTR             ; 2830
                                           50  D5 00068 2$:    TSTL    INDEX                                   ; 2832
                                           0C  13 0006A        BEQL    3$                                      ;
                                     0D    A0  9F 0006C        PUSHAB  13(INDEX)                               ; 2838
                          7E         0C    A0  9A 0006F        MOVZBL  12(INDEX), -(SP)                        ;
                                           55  DD 00073        PUSHL   R5                                      ;
                                         00EF  31 00075        BRW     13$                                     ;
                                           6E  D4 00078 3$:    CLRL    NT_PTR                                  ; 2849
                                           52  D4 0007A        CLRL    ARRAY_DESC_ADDR                         ; 2850
                          50               68  D0 0007C        MOVL    PAT$GB_MOD_PTR, R0                      ; 2855
                          03         02    A0  E8 0007F        BLBS    2(R0), 5$                               ;
                                         00E5  31 00083 4$:    BRW     14$                                     ;
                          F6 00000000G     EF  E9 00086 5$:    BLBC    PAT$GB_SYMBOLS, 4$                      ;
                                           7E  D4 0008D        CLRL    -(SP)                                   ; 2863
                                     04    AE  9F 0008F        PUSHAB  NT_PTR                                  ;
                                     04    AC  DD 00092        PUSHL   VALUE                                   ;
                00000000G EF               03  FB 00095        CALLS   #3, PAT$VAL_TO_SYM                      ;
                          E4               50  E9 0009C        BLBC    R0, 4$                                  ;
                                     04    AE  9F 0009F        PUSHAB  REAL_VALUE                              ; 2870
                                     04    AE  DD 000A2        PUSHL   NT_PTR                                  ;
                00000000G EF               02  FB 000A5        CALLS   #2, PAT$SYMBOL_VALU                     ;
                          D4               50  E9 000AC        BLBC    R0, 4$                                  ;
           50             6E               69  C1 000AF        ADDL3   PAT$GL_RST_BEGN, NT_PTR, R0             ; 2880
                                     08    A0  B5 000B3        TSTW    8(R0)                                   ;
                                           3E  13 000B6        BEQL    8$                                      ;
                          50         04    A0  D0 000B8        MOVL    4(R0), DST_RECRD                        ; 2891
 02      02   A0          02               00  ED 000BC        CMPZV   #0, #2, 2(DST_RECRD), #2                ; 2892
                                           32  12 000C2        BNEQ    8$                                      ;
 02      02   A0          02               02  ED 000C4        CMPZV   #2, #2, 2(DST_RECRD), #2                ; 2893
                                           08  12 000CA        BNEQ    6$                                      ;
 0F      02   A0          04               04  ED 000CC        CMPZV   #4, #4, 2(DST_RECRD), #15               ; 2894
                                           0D  13 000D2        BEQL    7$                                      ;
                          0C         02    A0  93 000D4 6$:    BITB    2(DST_RECRD), #12                       ; 2895
                                           1C  12 000D8        BNEQ    8$                                      ;
                     F0   8F         02    A0  93 000DA        BITB    2(DST_RECRD), #240                      ; 2896
                                           15  12 000DF        BNEQ    8$                                      ;
                          52         04    AE  D0 000E1 7$:    MOVL    REAL_VALUE, ARRAY_DESC_ADDR             ; 2905
                                     08    AE  9F 000E5        PUSHAB  BOUNDS                                  ; 2906
                                           52  DD 000E8        PUSHL   ARRAY_DESC_ADDR                         ;
                00000000G EF               02  FB 000EA        CALLS   #2, PAT$GET_BOUNDS                      ;
                       04 AE         08    AE  D0 000F1        MOVL    BOUNDS, REAL_VALUE                      ; 2907
           50          04 AC         04    AE  C3 000F6 8$:    SUBL3   REAL_VALUE, VALUE, R0                   ; 2916
```

```
              00000100  8F          50 D1 000FC        CMPL    R0, #256                        ;
                                    09 1F 00103        BLSSU   9$                              ;
        50              6E          69 C1 00105        ADDL3   PAT$GL_RST_BEGN, NT_PTR, R0     ; 2917
                               03   A0 95 00109        TSTB    3(R0)                           ;
                                    5D 18 0010C        BGEQ    14$                             ;
                               10   AE 9F 0010E 9$:    PUSHAB  PATH_VEC                        ; 2928
                               04   AE DD 00111        PUSHL   NT_PTR                          ;
              00000000G EF          02 FB 00114        CALLS   #2, PAT$ADD_NT_T_PV             ;
                               10   AE 9F 0011B        PUSHAB  PATH_VEC                        ; 2933
              00000000V EF          01 FB 0011E        CALLS   #1, PAT$PRINT_PATH              ;
                     04 AE     04   AC D1 00125        CMPL    VALUE, REAL_VALUE               ; 2940
                                    4D 13 0012A        BEQL    15$                             ;
                               04   A5 9F 0012C        PUSHAB  P.AAC                           ; 2949
                        66          01 FB 0012F        CALLS   #1, PAT$FAO_PUT                 ;
                     04 AC     04   AE C2 00132        SUBL2   REAL_VALUE, VALUE               ; 2950
                        10     00   B8 91 00137        CMPB    @PAT$GB_MOD_PTR, #16            ; 2959
                                    2E 12 0013B        BNEQ    14$                             ;
                        50          07 D0 0013D        MOVL    #7, I                           ; 2974
                        52     01   A0 9E 00140 10$:   MOVAB   1(R0), CLUDGE_SIZE              ; 2977
        51              50          02 78 00144        ASHL    #2, I, R1                       ; 2978
  53         04 AC              04  51 EF 00148        EXTZV   R1, #4, VALUE, CLUDGE_VAL       ;
                                    04 12 0014E        BNEQ    11$                             ;
                                    50 D7 00150        DECL    I                               ; 2974
                                    EC 11 00152        BRB     10$                             ;
                        0A          53 D1 00154 11$:   CMPL    CLUDGE_VAL, #10                 ; 2983
                                    06 1F 00157        BLSSU   12$                             ;
                               08   A5 9F 00159        PUSHAB  P.AAD                           ; 2985
                        66          01 FB 0015C        CALLS   #1, PAT$FAO_PUT                 ;
                               04   AC DD 0015F 12$:   PUSHL   VALUE                           ; 2987
                                    52 DD 00162        PUSHL   CLUDGE_SIZE                     ;
                               0C   A5 9F 00164        PUSHAB  P.AAE                           ; 2986
                        66          03 FB 00167 13$:   CALLS   #3, PAT$FAO_PUT                 ;
                                       04 0016A        RET                                     ; 2961
                                    01 DD 0016B 14$:   PUSHL   #1                              ; 2997
                        7E     08   AC 7D 0016D        MOVQ    NEW_LENGTH, -(SP)               ;
                               04   AC DD 00171        PUSHL   VALUE                           ;
                   008B C6          04 FB 00174        CALLS   #4, PAT$OUT_NUM_VAL             ;
                                       04 00179 15$:   RET                                     ; 2998

; Routine Size:  378 bytes,    Routine Base:  _PAT$CODE + 0149
```

```
1 GLOBAL ROUTINE PAT$SHOW_MODULE : NOVALUE =
1
1 !++
1 ! FUNCTIONAL DESCRIPTION:
1 !
1 !       Show which modules from the MC are in the RST.
1 !
1 ! FORMAL PARAMETERS:
1 !
1 !       NONE.
1 !
1 ! IMPLICIT INPUTS:
1 !
1 !       PAT$GL_MC_PTR   -I s assumed to point to the beginning
1 !                        of the MC.
1 !
1 ! IMPLICIT OUTPUTS:
1 !
1 !       NONE
1 !
1 ! COMPLETION CODES:
1 !
1 !       NOVALUE.
1 !
1 ! SIDE EFFECTS:
1 !
1 !       A summary of the current modules which the RST knows
1 !       about is written to the console.
1 !
1 !--
1
2 BEGIN
2 LOCAL
2       OUTPUT_BUFFER : VECTOR [TTY_OUT_WIDTH, BYTE],        ! Buffer to build report in.
2       NUM_MC_ENTRIES,                                       ! How many entries there are in the MC.
2       MC_PTR : REF MC_RECORD;                               ! Pointer to current MC record.
2
2 !++
2 ! Set up to do standard PATCH I/O.
2 !--
2 PAT$CP_OUT_STR = OUTPUT_BUFFER +1;
2 PAT$GL_BUF_SIZ = 0;
2
2 !++
2 ! Initialize the pointer that we use to 'chain' thru the module chain.
2 !--
3 IF ((MC_PTR = .PAT$GL_MC_PTR[MC_NEXT]) EQL 0)
2 THEN
3       BEGIN
3       SIGNAL(PAT$_NOLCL+MSG$K_INFO);
3       RETURN;
2       END;
2
2 !++
2 ! Print out the title, and then loop thru the module chain simply giving the
2 ! relevant information for each module which we know about.
2 !--
```

```
2 NUM_MC_ENTRIES = 0;
2 PAT$FAO_PUT( UPLIT (%ASCIC 'module name  symbols  size!/'));
2 PAT$OUT_PUT(OUTPUT_BUFFER);
2 DO
3           BEGIN
3           !++
3           ! Print out the standard information - name and
3           ! whether or not it has been initialized into the RST.
3           !--
3           NUM_MC_ENTRIES = .NUM_MC_ENTRIES +1;
3           PAT$CP_OUT_STR = OUTPUT_BUFFER +1;
3           PAT$GL_BUF_SIZ = 0;
3           PAT$FAO_PUT( UPLIT( %ASCIC '!15AC !3AC   ' ), MC_PTR [MC_NAME_CS],
4                       (IF (.MC_PTR [MC_IN_RST]) THEN UPLIT( %ASCIC 'yes' )
3                                                 ELSE UPLIT( %ASCIC 'no' )));
3
3           !++
3           ! We have to call a routine to output the size
3           ! so that it comes out in the right RADIX.
3           !--
3           PAT$FAO_PUT(UPLIT(%ASCIC'!UL.'),PAT$MODULE_SIZE(.MC_PTR));
3           PAT$OUT_PUT(OUTPUT_BUFFER);                         ! Finally output the buffer.
3           END
2 WHILE ((MC_PTR = .MC_PTR[MC_NEXT]) NEQ 0);
2
2 !++
2 ! Give final summary information.
2 !--
2 PAT$CP_OUT_STR = OUTPUT_BUFFER +1;
2 PAT$GL_BUF_SIZ = 0;
2 PAT$FAO_PUT( UPLIT( %ASCIC '!/total modules:  !UL.'), .NUM_MC_ENTRIES);
2 PAT$FAO_PUT( UPLIT( %ASCIC '!/remaining size: !UL.'), PAT$REPORT_FREE());
2 PAT$OUT_PUT(OUTPUT_BUFFER);
1 END;


                                                                  .PSECT  _PAT$PLIT,NOWRT,NOEXE,0

73 20 20 65 6D 61 6E 20 65 6C 75 64 6F 6D 1C 00018 P.AAF:  .ASCII  <28>\module name  symbols  size!/\<0>    ;
00 2F 21 65 7A 69 73 20 20 73 6C 6F 62 6D 79 00027                                                  ;
                                           00 00 00036          .ASCII  <0><0>                      ;
00 20 20 20 43 41 33 21 20 43 41 35 31 21 0D 00038 P.AAG:  .ASCII  <13>\!15AC !3AC   \<0><0>        ;
                                              00 00047                                              ;
                                     73 65 79 03 00048 P.AAH:  .ASCII  <3>\yes\                     ;
                                     00 6F 6E 02 0004C P.AAI:  .ASCII  <2>\no\<0>                   ;
                      00 00 00 2E 4C 55 21 04 00050 P.AAJ:  .ASCII  <4>\!UL.\<0><0><0>             ;
65 6C 75 64 6F 6D 20 6C 61 74 6F 74 2F 21 16 00058 P.AAK:  .ASCII  <22>\!/total modules:  !UL.\<0> ;
                  00 2E 4C 55 21 20 20 3A 73 00067                                                  ;
69 73 20 67 6E 69 6E 69 61 6D 65 72 2F 21 16 00070 P.AAL:  .ASCII  <22>\!/remaining size: !UL.\<0> ;
                  00 2E 4C 55 21 20 3A 65 7A 0007F                                                  ;


                                                                  .PSECT  _PAT$CODE,NOWRT,2

                                          01FC 00000          .ENTRY  PAT$SHOW_MODULE, Save R2,R3,R4,R5,R6,R7,R8  ; 2999
```

```
                58 00000000G  EF  9E 00002      MOVAB   PAT$CP_OUT_STR, R8                      ;
                57 00000000G  EF  9E 00009      MOVAB   PAT$GL_RST_BEGN, R7                     ;
                56 00000000G  EF  9E 00010      MOVAB   PAT$GL_BUF_SIZ, R6                      ;
                55 00000000'  EF  9E 00017      MOVAB   P.AAF, R5                               ;
                54      FD1B  CF  9E 0001E      MOVAB   PAT$FAO_PUT, R4                         ;
                5E      FF7C  CE  9E 00023      MOVAB   -132(SP), SP                            ;
                68        01  AE  9E 00028      MOVAB   OUTPUT_BUFFER+1, PAT$CP_OUT_STR         ; 3039
                              66  D4 0002C      CLRL    PAT$GL_BUF_SIZ                          ; 3040
      50 00000000G  EF        67  C1 0002E      ADDL3   PAT$GL_RST_BEGN, PAT$GL_MC_PTR, R0      ; 3045
                    52        60  3C 00036      MOVZWL  (R0), MC_PTR                            ;
                              0E  12 00039      BNEQ    1$                                      ;
                006D81CB      8F  DD 0003B      PUSHL   #7176651                                ; 3048
       00000000G  00          01  FB 00041      CALLS   #1, LIB$SIGNAL                          ;
                              04  00048         RET                                             ; 3047
                              53  D4 00049 1$:  CLRL    NUM_MC_ENTRIES                          ; 3056
                              55  DD 0004B      PUSHL   R5                                      ; 3057
                64            01  FB 0004D      CALLS   #1, PAT$FAO_PUT                         ;
                              5E  DD 00050      PUSHL   SP                                      ; 3058
            0128  C4          01  FB 00052      CALLS   #1, PAT$OUT_PUT                         ;
                              53  D6 00057 2$:  INCL    NUM_MC_ENTRIES                          ; 3065
                68        01  AE  9E 00059      MOVAB   OUTPUT_BUFFER+1, PAT$CP_OUT_STR         ; 3066
                              66  D4 0005D      CLRL    PAT$GL_BUF_SIZ                          ; 3067
      50            52        67  C1 0005F      ADDL3   PAT$GL_RST_BEGN, MC_PTR, R0             ; 3069
      06       03   A0        01  E1 00063      BBC     #1, 3(R0), 3$                           ;
                51        30  A5  9E 00068      MOVAB   P.AAH, R1                               ;
                              04  11 0006C      BRB     4$                                      ;
                51        34  A5  9E 0006E 3$:  MOVAB   P.AAI, R1                               ; 3070
                              51  DD 00072 4$:  PUSHL   R1                                      ;
                          0C  A0  9F 00074      PUSHAB  12(R0)                                  ; 3068
                          20  A5  9F 00077      PUSHAB  P.AAG                                   ;
                64            03  FB 0007A      CALLS   #3, PAT$FAO_PUT                         ;
                              52  DD 0007D      PUSHL   MC_PTR                                  ; 3076
              4B  A4          01  FB 0007F      CALLS   #1, PAT$MODULE_SIZE                     ;
                              50  DD 00083      PUSHL   R0                                      ;
                          38  A5  9F 00085      PUSHAB  P.AAJ                                   ;
                64            02  FB 00088      CALLS   #2, PAT$FAO_PUT                         ;
                              5E  DD 0008B      PUSHL   SP                                      ; 3077
            0128  C4          01  FB 0008D      CALLS   #1, PAT$OUT_PUT                         ;
      50            52        67  C1 00092      ADDL3   PAT$GL_RST_BEGN, MC_PTR, R0             ; 3079
                    52        60  3C 00096      MOVZWL  (R0), MC_PTR                            ;
                              BC  12 00099      BNEQ    2$                                      ;
                68        01  AE  9E 0009B      MOVAB   OUTPUT_BUFFER+1, PAT$CP_OUT_STR         ; 3084
                              66  D4 0009F      CLRL    PAT$GL_BUF_SIZ                          ; 3085
                              53  DD 000A1      PUSHL   NUM_MC_ENTRIES                          ; 3086
                          40  A5  9F 000A3      PUSHAB  P.AAK                                   ;
                64            02  FB 000A6      CALLS   #2, PAT$FAO_PUT                         ;
       00000000G  EF          00  FB 000A9      CALLS   #0, PAT$REPORT_FREE                     ; 3087
                              50  DD 000B0      PUSHL   R0                                      ;
                          58  A5  9F 000B2      PUSHAB  P.AAL                                   ;
                64            02  FB 000B5      CALLS   #2, PAT$FAO_PUT                         ;
                              5E  DD 000B8      PUSHL   SP                                      ; 3088
            0128  C4          01  FB 000BA      CALLS   #1, PAT$OUT_PUT                         ;
                              04  000BF         RET                                             ; 3089

; Routine Size:  192 bytes,    Routine Base:  _PAT$CODE + 02C3
```

```
1 GLOBAL ROUTINE PAT$SHOW_SCOPE( PATH_VEC_PTR ) : NOVALUE =
1
1 !++
1 ! Functional Description:
1 !
1 !       Print out the current scope position vector (CSP).
1 !
1 ! Formal Parameters:
1 !
1 !       none.
1 !
1 ! Implicit Inputs:
1 !
1 !       The CSP is a PATHNAME_VECTOR.
1 !
1 ! Return Value:
1 !
1 !       NOVALUE
1 !
1 !--
1
2 BEGIN
2
2 LOCAL
2       OUTPUT_BUFFER : VECTOR [TTY_OUT_WIDTH, BYTE];          ! Build output message here.
2
2 !++
2 ! Set up to use a new output buffer, and encode the standard beginning of
2 ! the SHOW SCOPE message into it.
2 !--
2 PAT$CP_OUT_STR = OUTPUT_BUFFER + 1;
2 PAT$GL_BUF_SIZ = 0;
2
2 PAT$FAO_PUT( UPLIT( %ASCIC 'scope: ') );
2
2 !++
2 ! The actual thing we print out depends on whether or not there is actually
2 ! a CSP just now.
2 !--
3 IF (.PAT$GL_CSP_PTR NEQ 0)
2 THEN
2       PAT$PRINT_PATH( .PAT$GL_CSP_PTR )
2 ELSE
2       PAT$FAO_PUT( UPLIT( %ASCIC '<null>' ));
2
2 !++
2 ! And finally force out the message.
2 !--
2 PAT$OUT_PUT( OUTPUT_BUFFER );
1 END;


                                                       .PSECT  _PAT$PLIT,NOWRT,NOEXE,0

        20  3A  65  70  6F  63  73  07   00088 P.AAM:  .ASCII  <7>\scope: \                     ;
        00  3E  6C  6C  75  6E  3C  06   00090 P.AAN:  .ASCII  <6>\<null>\<0>                   ;
```

```
                                                  .PSECT  _PAT$CODE,NOWRT,2

                                 0004 00000       .ENTRY  PAT$SHOW_SCOPE, Save R2                    ; 3090
               52     FC77   CF   9E 00002        MOVAB   PAT$FAO_PUT, R2
               5E     FF7C   CE   9E 00007        MOVAB   -132(SP), SP
     00000000G EF       01   AE   9E 0000C        MOVAB   OUTPUT_BUFFER+1, PAT$CP_OUT_STR            ; 3120
                 00000000G   EF   D4 00014        CLRL    PAT$GL_BUF_SIZ                             ; 3121
                 00000000'   EF   9F 0001A        PUSHAB  P.AAM                                      ; 3123
               62            01   FB 00020        CALLS   #1, PAT$FAO_PUT
               50 00000000G  EF   D0 00023        MOVL    PAT$GL_CSP_PTR, R0                         ; 3129
                             0B   13 0002A        BEQL    1$
                             50   DD 0002C        PUSHL   R0                                         ; 3131
     00000000V EF            01   FB 0002E        CALLS   #1, PAT$PRINT_PATH
                             09   11 00035        BRB     2$
                 00000000'   EF   9F 00037 1$:    PUSHAB  P.AAN                                      ; 3133
               62            01   FB 0003D        CALLS   #1, PAT$FAO_PUT
                             5E   DD 00040 2$:    PUSHL   SP                                         ; 3138
          0128 C2            01   FB 00042        CALLS   #1, PAT$OUT_PUT
                                  04 00047        RET                                                ; 3139

; Routine Size:  72 bytes,    Routine Base:  _PAT$CODE + 0383
```

```
1 GLOBAL ROUTINE PAT$PV_TO_CS( PATH_VEC_PTR, OUTPUT_BUF ) : NOVALUE =
1
1 !++
1 ! Functional Description:
1 !
1 !       Encode a pathname vector into a given vector
1 !       so that what we have built is a counted string.
1 !
1 ! Formal Parameters:
1 !
1 !       PATH_VEC_PTR    -A pointer to the pathname vector we
1 !                        want encoded.
1 !
1 !       OUTPUT_BUF      -A pointer to where we build the counted string.
1 !
1 ! Implicit Inputs:
1 !
1 !       We will encode up to TTY_OUTPUT_WIDTH characters.
1 !       The rest of the pathname vector is ignored.
1 !
1 ! Return Value:
1 !
1 !       NOVALUE.
1 !
1 ! Side Effects:
1 !
1 !       The pathname is encoded into the given buffer.
1 !
1 !--
1
2 BEGIN
2
2 MAP
2       OUTPUT_BUF : REF VECTOR[,BYTE],
2       PATH_VEC_PTR : REF PATHNAME_VECTOR;
2
2 LOCAL
2       CHAR_COUNT,
2       OUTPUT_PTR : REF VECTOR[,BYTE];
2
2 !++
2 ! Make sure we are called with valid parameters.
2 !--
3 IF (.PATH_VEC_PTR EQL 0) OR (.OUTPUT_BUF EQL 0)
2 THEN
2       RETURN;
2
2 !++
2 ! The size of the counted string we build starts off as 0.  The first element
2 ! in the string starts at the next character.
2 !--
2 CHAR_COUNT = 0;
2 OUTPUT_PTR = OUTPUT_BUF[1];
2
2 !++
2 ! Just encode the characters into the indicated position in the buffer until we
2 ! have exhausted the CS pointers in the pathname vector.
```

```
2 !--
2 INCR I FROM 0 TO MAX_PATH_SIZE
2         DO
3         BEGIN
3         LOCAL
3                 CS_PTR : CS_POINTER;                       ! Each element of the pathname vector
3                                                            ! is a pointer to a counted string.
3
3         !++
3         ! Terminate when the path vector ends.
3         !--
4         IF ((CS_PTR = .PATH_VEC_PTR[.I]) EQL 0)
3         THEN
3                 EXITLOOP;
3
3         !++
3         ! Output the pathname element separation character in all but the first
3         ! iteration.
3         !--
4         IF (.I NEQ 0)
3         THEN
4                 BEGIN
4                 CHAR_COUNT = .CHAR_COUNT +1;
4                 OUTPUT_PTR[0] = %C'\';
4                 OUTPUT_PTR = .OUTPUT_PTR + 1;
3                 END;
3
3         !++
3         ! See if we have enough room left for the next pathname element.
3         !--
4         IF (NOT .CHAR_COUNT + .CS_PTR[0] LSS TTY_OUT_WIDTH)
3         THEN
3                 EXITLOOP;
3         CH$MOVE( .CS_PTR[0], CS_PTR[1], .OUTPUT_PTR );
3         CHAR_COUNT = .CHAR_COUNT +.CS_PTR[0];
3         OUTPUT_PTR = .OUTPUT_PTR + .CS_PTR[0];
2         END;
2
2 !++
2 ! Now make the output buffer a real counted string.
2 !--
2 OUTPUT_BUF[0] = .CHAR_COUNT;
1 END;



                              03FC 00000        .ENTRY  PAT$PV_TO_CS, Save R2,R3,R4,R5,R6,R7,R8,R9   ; 3140
                   04    AC    D5 00002         TSTL    PATH_VEC_PTR                                   ; 3183
                         48    13 00005         BEQL    4$                                             ;
                   08    AC    D5 00007         TSTL    OUTPUT_BUF                                     ;
                         43    13 0000A         BEQL    4$                                             ;
     59       08   AC    01    C1 0000C         ADDL3   #1, OUTPUT_BUF, OUTPUT_PTR                     ; 3192
                         56    7C 00011         CLRQ    CHAR_COUNT                                     ; 3191
           58      04 BC47     D0 00013 1$:     MOVL    @PATH_VEC_PTR[I], CS_PTR                       ; 3208
                         31    13 00018         BEQL    3$                                             ;
```

```
                                     57  D5 0001A          TSTL    I                                       ; 3216
                                     06  13 0001C          BEQL    2$                                      ;
                                     56  D6 0001E          INCL    CHAR_COUNT                              ; 3219
                  89         5C      8F  90 00020          MOVB    #92, (OUTPUT_PTR)+                      ; 3220
                  50                 68  9A 00024 2$:      MOVZBL  (CS_PTR), R0                            ; 3227
                  50                 56  C0 00027          ADDL2   CHAR_COUNT, R0                          ;
         00000084 8F                 50  D1 0002A          CMPL    R0, #132                                ;
                  18                 18  00031             BGEQ    3$                                      ;
                  50                 68  9A 00033          MOVZBL  (CS_PTR), R0                            ; 3230
     69       01  A8                 50  28 00036          MOVC3   R0, 1(CS_PTR), (OUTPUT_PTR)             ;
                  50                 68  9A 0003B          MOVZBL  (CS_PTR), R0                            ; 3231
                  56                 50  C0 0003E          ADDL2   R0, CHAR_COUNT                          ;
                  50                 68  9A 00041          MOVZBL  (CS_PTR), R0                            ; 3232
                  59                 50  C0 00044          ADDL2   R0, OUTPUT_PTR                          ;
     C8           57                 0A  F3 00047          AOBLEQ  #10, I, 1$                              ; 3198
              08  BC                 56  90 0004B 3$:      MOVB    CHAR_COUNT, @OUTPUT_BUF                 ; 3238
                                         04 0004F 4$:      RET                                             ; 3239

; Routine Size:  80 bytes,    Routine Base:  _PAT$CODE + 03CB
```

```
1 GLOBAL ROUTINE PAT$PRINT_PATH( PATH_VEC_PTR ) : NOVALUE =
1
1 !++
1 ! Functional Description:
1 !
1 !       Encode a pathname vector into the output stream
1 !
1 ! Formal Parameters:
1 !
1 !       PATH_VEC_PTR    -A pointer to the pathname vector we want printed.
1 !
1 ! Implicit Inputs:
1 !
1 !       Some exterior routine has done all the setup for
1 !       PATCH standard I/O.  This routine just does 'puts'.
1 !
1 ! Return Value:
1 !
1 !       NOVALUE.
1 !
1 ! Side Effects:
1 !
1 !       The pathname is encoded into the current output buffer.
1 !
1 !--
1
2 BEGIN
2 MAP
2       PATH_VEC_PTR : REF PATHNAME_VECTOR;
2
2 LOCAL
2       SYM_BUFFER : VECTOR[ TTY_OUT_WIDTH, BYTE];
2
2 !++
2 ! First encode the characters into our own buffer - because we have a routine
2 ! that already knows how to do this.
2 !--
2 PAT$PV_TO_CS( .PATH_VEC_PTR, SYM_BUFFER );
2
2 !++
2 ! Then simply output the pathname.
2 !--
2 PAT$FAO_PUT(UPLIT(%ASCIC '!AC'), SYM_BUFFER );
1 END;


                                          .PSECT  _PAT$PLIT,NOWRT,NOEXE,0

              43  41  21  03  00098 P.AA0: .ASCII  <3>\!AC\                               ;


                                          .PSECT  _PAT$CODE,NOWRT,2

                        0000 00000        .ENTRY  PAT$PRINT_PATH, Save nothing             ; 3240
     5E      FF7C   CE   9E 00002         MOVAB   -132(SP), SP                             ;
```

```
                                  5E  DD 00007          PUSHL   SP                            ; 3277
                          04      AC  DD 00009          PUSHL   PATH_VEC_PTR                  ;
          A0   AF                 02  FB 0000C          CALLS   #2, PAT$PV_TO_CS              ;
                                  5E  DD 00010          PUSHL   SP                            ; 3282
                   00000000'      EF  9F 00012          PUSHAB  P.AAO                         ;
        FBC8   CF                 02  FB 00018          CALLS   #2, PAT$FAO_PUT               ;
                                      04 0001D          RET                                   ; 3283

; Routine Size:  30 bytes,    Routine Base:  _PAT$CODE + 041B
```

```
; 1039          3284  1 END
; 1040          3285  0 ELUDOM

                                                        .EXTRN  LIB$SIGNAL

;                          PSECT SUMMARY
;
;       Name                       Bytes                     Attributes
;
;    _PAT$CODE                     1081  NOVEC,NOWRT,  RD ,  EXE,NOSHR,  LCL,  REL,  CON,NOPIC,ALIGN(2)
;    .  ABS  .                        0  NOVEC,NOWRT,NORD ,NOEXE,NOSHR,  LCL,  ABS,  CON,NOPIC,ALIGN(0)
;    _PAT$OWN                         4  NOVEC,  WRT,  RD ,NOEXE,NOSHR,  LCL,  REL,  CON,NOPIC,ALIGN(2)
;    _PAT$PLIT                      156  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        00:01.9



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

; Size:          1081 code + 160 data bytes
; Run Time:         00:35.3
; Elapsed Time:     01:56.3
; Lines/CPU Min:     5581
; Lexemes/CPU-Min: 34161
; Memory Used:  227 pages
; Compilation Complete
```

0303 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

PATREB
LIS

PATSCA
LIS

PATSIO
LIS

PATRST
LIS

PATSPA
LIS

PATSSV
LIS

0304 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

PHONE

PATSYM
LIS

PHONE
MAP

BASICCMDS
LIS

PATSTO
LIS

PATVEC
LIS

PHONEREQ
REQ

PATWRT
LIS

FILECMDS
LIS