```
FFFFFFFFFFFFFFF      AAAAAAAAA     LLL
FFFFFFFFFFFFFFF      AAAAAAAAA     LLL
FFFFFFFFFFFFFFF      AAAAAAAAA     LLL
FFF               AAA         AAA  LLL
FFF               AAA         AAA  LLL
FFF               AAA         AAA  LLL
FFF               AAA         AAA  LLL
FFF               AAA         AAA  LLL
FFF               AAA         AAA  LLL
FFFFFFFFFFFF      AAA         AAA  LLL
FFFFFFFFFFFF      AAA         AAA  LLL
FFFFFFFFFFFF      AAA         AAA  LLL
FFF               AAAAAAAAAAAAAAA  LLL
FFF               AAAAAAAAAAAAAAA  LLL
FFF               AAAAAAAAAAAAAAA  LLL
FFF               AAA         AAA  LLL
FFF               AAA         AAA  LLL
FFF               AAA         AAA  LLL
FFF               AAA         AAA  LLLLLLLLLLLLLLL
FFF               AAA         AAA  LLLLLLLLLLLLLLL
FFF               AAA         AAA  LLLLLLLLLLLLLLL
```

FILEID**FALACTION

```
FFFFFFFFFF    AAAAAA    LL            AAAAAA      CCCCCCCC  TTTTTTTTTT    IIIIII      OOOOOO    NN      NN
FFFFFFFFFF    AAAAAA    LL            AAAAAA      CCCCCCCC  TTTTTTTTTT    IIIIII      OOOOOO    NN      NN
FF          AA      AA  LL          AA      AA  CC              TT          II      OO      OO  NN      NN
FF          AA      AA  LL          AA      AA  CC              TT          II      OO      OO  NN      NN
FF          AA      AA  LL          AA      AA  CC              TT          II      OO      OO  NNNN    NN
FF          AA      AA  LL          AA      AA  CC              TT          II      OO      OO  NNNN    NN
FFFFFFFF    AA      AA  LL          AA      AA  CC              TT          II      OO      OO  NN  NN  NN
FFFFFFFF    AA      AA  LL          AA      AA  CC              TT          II      OO      OO  NN  NN  NN
FF          AAAAAAAAAA  LL          AAAAAAAAAA  CC              TT          II      OO      OO  NN    NNNN
FF          AAAAAAAAAA  LL          AAAAAAAAAA  CC              TT          II      OO      OO  NN    NNNN
FF          AA      AA  LL          AA      AA  CC              TT          II      OO      OO  NN      NN
FF          AA      AA  LL          AA      AA  CC              TT          II      OO      OO  NN      NN  ....
FF          AA      AA  LLLLLLLLLL  AA      AA    CCCCCCCC      TT        IIIIII      OOOOOO    NN      NN  ....
FF          AA      AA  LLLLLLLLLL  AA      AA    CCCCCCCC      TT        IIIIII      OOOOOO    NN      NN  ....
```

```
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
```

FAL
V04

Table of contents

FAL
V04

```
0000        .TITLE  FALACTION - STATE TABLE ACTION ROUTINES
0000        .IDENT  'V04-000'
0000
0000   ;
0000   ;****************************************************************************
0000   ;*                                                                          *
0000   ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
0000   ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
0000   ;*  ALL RIGHTS RESERVED.                                                    *
0000   ;*                                                                          *
0000   ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
0000   ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
0000   ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
0000   ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
0000   ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
0000   ;*  TRANSFERRED.                                                            *
0000   ;*                                                                          *
0000   ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
0000   ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
0000   ;*  CORPORATION.                                                            *
0000   ;*                                                                          *
0000   ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
0000   ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
0000   ;*                                                                          *
0000   ;*                                                                          *
0000   ;****************************************************************************
0000   ;
0000
0000   ;++
0000   ; Facility: FAL (DECnet File Access Listener)
0000   ;
0000   ; Abstract:
0000   ;
0000   ;       This module contains action routines called by the state table manager.
0000   ;
0000   ; Environment: VAX/VMS, user mode
0000   ;
0000   ; Author: James A. Krycka,      Creation Date:  16-JUN-1977
0000   ;
0000   ; Modified By:
0000   ;
0000   ;       V03-020 JEJ0052         J E Johnson     21-Aug-1984
0000   ;               Fix the sharing support in the NT$CREATE logic to not
0000   ;               indiscriminately turn off the SHRGET bit.  Note that
0000   ;               since some non-VMS always set SHRGET on $CREATE this bit
0000   ;               is turned off if the file format does not support sharing.
0000   ;
0000   ;       V03-019 JEJ0040         J E Johnson     19-Jun-1984
0000   ;               Fix 3-part name message handling to always send the
0000   ;               directory spec if the device spec has changed.
0000   ;
0000   ;       V03-018 JAK0145         J A Krycka      12-APR-1984
0000   ;               Track changes in DAP message building algorithm.
0000   ;
0000   ;       V03-017 JAK0143         J A Krycka      11-APR-1984
0000   ;               Use NAM$V_SRCHXABS option of the NAM$B_NOP field instead of the
0000   ;               FAB$L_CTX field to request return of XAB information on $SEARCH.
```

```
0000  ;
0000  ;       V03-016 JAK0137          J A Krycka       12-MAR-1984
0000  ;               Implement FAL logging option to disable poor-man's (or manual)
0000  ;               routing which is an unsupported and undocumented feature.
0000  ;
0000  ;       V03-015 JAK0136          J A Krycka       07-MAR-1984
0000  ;               Use FAL$GQ_WILDSPEC for wildcard file specification string.
0000  ;
0000  ;       V03-014 JAK0129          J A Krycka       11-JAN-1984
0000  ;               Correct success exit path in WRITE_FTM_NORBK action routine.
0000  ;
0000  ;       V03-013 JAK0128          J A Krycka       16-SEP-1983
0000  ;               Add FAL$DISCARD_DAT action routine to CRC check data being
0000  ;               discarded during file transfer storage mode error recovery.
0000  ;
0000  ;       V03-012 JAK0120          J A Krycka       29-JUL-1983
0000  ;               Add FAL$SAVE_MSG action routine to defer message processing.
0000  ;               Add FAL$BIT_BUCKET action routine to discard buffered data
0000  ;               during file transfer storage mode error recovery.
0000  ;               Also, reclassify error on Access Complete (end-of-stream) as
0000  ;               a file close error instead of a file transfer error to enable
0000  ;               the accessing system to distinguish between a $PUT/$WRITE
0000  ;               and a $DISCONNECT failure in file transfer storage mode.
0000  ;
0000  ;       V03-011 JAK0117          J A Krycka       08-JUL-1983
0000  ;               Preserve alternate success codes on record mode operations
0000  ;               instead of converting them to RMS$_SUC.
0000  ;
0000  ;       V03-010 JAK0110          J A Krycka       22-JUN-1983
0000  ;               Fix bug in return of Acknowledge message in FAL$DIR_SEARCH.
0000  ;
0000  ;       V03-009 JAK0109          J A Krycka       20-JUN-1983
0000  ;               Change SEND_CMP to be a subroutine.
0000  ;               Rename SEND_STS_SUC to EXIT_STS_SUC and SEND_STS_FAIL to
0000  ;               EXIT_STS_FAIL, as they also terminate the action routine.
0000  ;               Also, miscellaneous cleanup.
0000  ;
0000  ;       V03-008 JAK0108          J A Krycka       16-JUN-1983
0000  ;               Fix bug in CRC computation for retrieval of VFC formatted file
0000  ;               introduced in JAK0104.
0000  ;
0000  ;       V03-007 KRM0112          K Malik          31-MAY-1983
0000  ;               Continuation of KRM0106 to support DAP V7.0 specification.
0000  ;
0000  ;       V03-006 KRM0106          K Malik          10-MAY-1983
0000  ;               Update FAL$ERASE, FAL$EXTEND, and FAL$DIR_SEARCH to support
0000  ;               DAP V7.0 message exchange rules.
0000  ;
0000  ;       V03-005 JAK0104          J A Krycka       19-APR-1983
0000  ;               Rewrite FAL$GET_READ and FAL$PUT_WRITE action routines to
0000  ;               improve block I/O file transfer mode performance and to
0000  ;               organize these routines into eight distinct code paths for
0000  ;               clarity and efficiency. The FAL$GET_READ and FAL$PUT_WRITE
0000  ;               state table entry point names have been replaced by
0000  ;               FAL$RETRV_RAM, FAL$RETRV_FTM, FAL$STORE_RAM, and
0000  ;               FAL$STORE_FTM plus a new one called FAL$STORE_END.
0000  ;
```

```
0000    ;       V03-004 KRM0095         K Malik         06-APR-1983
0000    ;               Update FAL$RENAME to support DAP V7.0 specification.
0000    ;               Also, change SEND_ACK to be a subroutine.
0000    ;
0000    ;       V03-003 KRM0082         K Malik         23-MAR-1983
0000    ;               Add support for STMLF and STMCR file formats.
0000    ;
0000    ;       V03-002 KRM0067         K Malik         23-NOV-1982
0000    ;               Add FAL$RENAME routine to support $RENAME operation.
0000    ;
0000    ;--
```

```
            0000         .SBTTL  DECLARATIONS
        00000000         .PSECT  FAL$DATA        SHR,NOEXE,RD,WRT,LONG
            0000
            0000   ;
            0000   ; Include Files:
            0000   ;
            0000
            0000           $DAPPLGDEF                      ; Define DAP prologue symbols
            0000           $DAPHDRDEF                      ; Define DAP message header
            0000           $DAPCNFDEF                      ; Define DAP Configuration message
            0000           $DAPATTDEF                      ; Define DAP Attributes message
            0000           $DAPACCDEF                      ; Define DAP Access message
            0000           $DAPCTLDEF                      ; Define DAP Control message
            0000           $DAPCONDEF                      ; Define DAP Continue Transfer message
            0000           $DAPACKDEF                      ; Define DAP Acknowledge message
            0000           $DAPCMPDEF                      ; Define DAP Access Complete message
            0000           $DAPDATDEF                      ; Define DAP Data message
            0000           $DAPSTSDEF                      ; Define DAP Status message
            0000           $DAPKEYDEF                      ; Define DAP Key Definition message
            0000           $DAPALLDEF                      ; Define DAP Allocation message
            0000           $DAPSUMDEF                      ; Define DAP Summary message
            0000           $DAPTIMDEF                      ; Define DAP Date and Time message
            0000           $DAPPRODEF                      ; Define DAP Protection message
            0000           $DAPNAMDEF                      ; Define DAP Name message
            0000           $DAPCRCDEF                      ; Define DAP CRC checksum symbols
            0000           $DAPFIDDEF                      ; Define DAP field ID symbols
            0000           $FABDEF                         ; Define File Access Block symbols
            0000           $FALWRKDEF                      ; Define FAL Work Area symbols
            0000           $FALSTBDEF                      ; Define Statistics Block symbols
            0000           $NAMDEF                         ; Define Name Block symbols
            0000           $RABDEF                         ; Define Record Access Block symbols
            0000           $RMSDEF                         ; Define RMS completion codes
            0000           $SSDEF                          ; Define System Service status codes
            0000           $XABDEF                         ; Define symbols common to all XABs
            0000           $XABALLDEF                      ; Define Allocation XAB symbols
            0000           $XABDATDEF                      ; Define Date and Time XAB symbols
            0000           $XABFHCDEF                      ; Define File Header Char XAB symbols
            0000           $XABKEYDEF                      ; Define Key Definition XAB symbols
            0000           $XABPRODEF                      ; Define Protection XAB symbols
            0000           $XABRDTDEF                      ; Define Revision Date and Time symbols
            0000           $XABSUMDEF                      ; Define Summary XAB symbols
            0000
            0000   ;
            0000   ; Macros:
            0000   ;
            0000   ;       None
            0000   ;
            0000   ; Equated Symbols:
            0000   ;
            0000
0000000A    0000   LF=10                                   ; Line feed
0000000B    0000   VT=11                                   ; Vertical tab
0000000C    0000   FF=12                                   ; Form feed
0000000D    0000   CR=13                                   ; Carriage return
00000010    0000   DLE=16                                  ; Data link error  (Control-P)
00000011    0000   DC1=17                                  ; Device control 1 (Control-Q)
00000012    0000   DC2=18                                  ; Device control 2 (Control-R)
```

```
00000013  0000  DC3=19                                  ; Device control 3 (Control-S)
00000014  0000  DC4=20                                  ; Device control 4 (Control-T)
0000001A  0000  CTRLZ=26                                ; Control-Z
0000001B  0000  ESC=27                                  ; Escape
          0000
00000A0D  0000  CRLF=^X0A0D                             ; ASCII codes for CR and LF
          0000
          0000          ASSUME  DAP$Q_DCODE_FLG EQ 0
          0000          ASSUME  FAL$Q_FLG EQ 0
          0000
          0000  ;
          0000  ; Own Storage:
          0000  ;
          0000  ; The following bitmask represents the DAP terminator set for stream (STM)
          0000  ; format files where each bit position corresponds to an ASCII character code
          0000  ; for a single-character terminator. Note that the default terminator, CRLF,
          0000  ; is not expressed in this mask.
          0000  ;
          0000
          0000  FAL$STM_MASK::                          ; DAP stream terminator bitmask
0C1F1C00  0000          .LONG   <<1@LF> + <1@FF> + <1@VT> + <1@CTRLZ> + <1@ESC> + -
          0004                  <1@DLE> + <1@DC1> + <1@DC2> + <1@DC3> + <1@DC4>>
```

```
    0004      .SBTTL  STATE TABLE ACTION ROUTINES
00000000      .PSECT  FAL$CODE        NOSHR,EXE,RD,NOWRT,BYTE
    0000
    0000  ;++
    0000  ; Functional Description:
    0000  ;
    0000  ;       This module contains action routines invoked by the state table
    0000  ;       manager (FAL$STATE).
    0000  ;
    0000  ;       The input parameters and completion codes listed below are applicable
    0000  ;       for all of these action routines. Note that an action routine may use
    0000  ;       R0-R7 and AP without restoring them on exit. R0 on exit, however, must
    0000  ;       represent a status code to indicate success/failure of the routine or
    0000  ;       a true/false condition, as appropriate. This status code is used by
    0000  ;       the state table manager to advance to the next state.
    0000  ;
    0000  ; Calling Sequence:
    0000  ;
    0000  ;       BSBW    FAL$name
    0000  ;
    0000  ; Input Parameters:
    0000  ;
    0000  ;       R8      Address of FAL work area
    0000  ;       R9      Address of DAP control block
    0000  ;       R10     Address of FAB
    0000  ;       R11     Address of RAB
    0000  ;
    0000  ; Implicit Inputs:
    0000  ;
    0000  ;       None
    0000  ;
    0000  ; Output Parameters:
    0000  ;
    0000  ;       R0      Completion Code
    0000  ;       R1-R7   Destroyed
    0000  ;       AP      Destroyed
    0000  ;
    0000  ; Implicit Outputs:
    0000  ;
    0000  ;       None
    0000  ;
    0000  ; Completion Codes:
    0000  ;
    0000  ;       R0      1 = success; 0 = failure
    0000  ;
    0000  ; Side Effects:
    0000  ;
    0000  ;       None
    0000  ;
    0000  ;--
```

```
                       0000          .SBTTL  FAL$NEXT_MSG, FAL$INTE_MSG
                       0000
                       0000  ;++
                       0000  ; The following routines have separate entry points but share common code ...
                       0000  ;
                       0000  ; This routine obtains the next DAP message from partner (excluding interrupt
                       0000  ; messages). Then it syntax checks this message and stores the results in the
                       0000  ; DAP control block.
                       0000  ;--
                       0000
                       0000  FAL$NEXT_MSG::                                     ; Entry point
   03 68    15    E5   0000          BBCC    #FAL$V_MBXAST,(R8),10$     ; Branch if no mailbox message has
                       0004                                             ;  been received
         FFF9'    30   0004          BSBW    FAL$MBX_RCV_QIO            ; Ignore any interrupt message and
                       0007                                             ;  re-issue mailbox read with AST
         FFF6'    30   0007  10$:    BSBW    FAL$RECEIVE                ; Get next DAP message and parse it
                       000A  CHECK_PARSE:                               ; FAL$INTE_MSG branches here
    08 18 A9      E9   000A          BLBC    DAP$L_DCODE_STS(R9),10$    ; Branch on message parse failure
                       000E
                       000E  ;
                       000E  ; The DAP message parse was successful; use message type as the next state
                       000E  ; table value.
                       000E  ;
                       000E
         1A A9    90   000E          MOVB    DAP$B_DCODE_MSG(R9),-      ; Store new state transition value
         10 A8         0011                  FAL$B_VALUE(R8)            ;
          0CD0    31   0013          BRW     EXIT_SUCCESS               ; Exit state with success
                       0016
                       0016  ;
                       0016  ; Parse of DAP message failed; send Status message to partner.
                       0016  ;
                       0016
   50    19 A9    9A   0016  10$:    MOVZBL  DAP$B_DCODE_FID(R9),R0     ; Get ID of field in error
   51    1A A9    9A   001A          MOVZBL  DAP$B_DCODE_MSG(R9),R1     ; Get message type number
   52    1B A9    9A   001E          MOVZBL  DAP$B_DCODE_MAC(R9),R2     ; Get maccode error value
          0CA7    31   0022          BRW     EXIT_STS_FAIL              ; Return error in Status message
                       0025
                       0025  ;++
                       0025  ; This routine obtains the next DAP message from partner sent as an interrupt
                       0025  ; message. Then it syntax checks this message and stores the results in the
                       0025  ; DAP control block.
                       0025  ;--
                       0025
                       0025  FAL$INTE_MSG::                                     ; Entry point
   7E    08 A9    7D   0025          MOVQ    DAP$Q_MSG_BUF1(R9),-(SP)   ; Save decode context
   7E    10 A9    7D   0029          MOVQ    DAP$Q_MSG_BUF2(R9),-(SP)   ; "
         FFD0'    30   002D          BSBW    FAL$RECEIVE_MBX            ; Get next DAP message delivered as
                       0030                                             ;  an interrupt message and parse it
   10 A9     8E   7D   0030          MOVQ    (SP)+,DAP$Q_MSG_BUF2(R9)   ; Restore decode context
   08 A9     8E   7D   0034          MOVQ    (SP)+,DAP$Q_MSG_BUF1(R9)   ; "
            D0    11   0038          BRB     CHECK_PARSE                ; Join common code
```

```
                               003A         .SBTTL  FAL$TEST_MSG
                               003A
                               003A  ;++
                               003A  ; This routine determines whether or not a DAP message has been received from
                               003A  ; partner. The receive AST flag stored in the FAL work area is used to form
                               003A  ; the state transition status code.
                               003A  ;--
                               003A
                               003A  FAL$TEST_MSG::                                ; Entry point
50   68   01   11     EF       003A         EXTZV   #FAL$V_RCVAST,#1,(R8),R0; Form yes/no reply
                      05       003F         RSB                                   ; Exit state with status code in R0
                               0040
                               0040
                               0040         .SBTTL  FAL$SAVE_MSG
                               0040
                               0040  ;++
                               0040  ; This routine restores the input message descriptor in the DAP control block
                               0040  ; to its state before the last message was requested (via FAL$NEXT_MSG) and
                               0040  ; decoded. This allows a message to be parsed, examined, and then returned
                               0040  ; to the input stream (via FAL$SAVE_MSG) to defer its processing (until after
                               0040  ; an interrupt message, for example, has been received and processed). To
                               0040  ; reparse the saved message, FAL$NEXT_MSG is called.
                               0040  ;--
                               0040
                               0040  FAL$SAVE_MSG::                                ; Entry point
               10 A9  C0       0040         ADDL2   DAP$Q_MSG_BUF2(R9),-          ; Restore size of input message
               08 A9           0043                 DAP$Q_MSG_BUF1(R9)            ;  descriptor
               14 A9  D0       0045         MOVL    DAP$Q_MSG_BUF2+4(R9),-        ; Restore address of input message
               0C A9           0048                 DAP$Q_MSG_BUF1+4(R9)          ;  descriptor
                0C99  31       004A         BRW     EXIT_SUCCESS                  ; Exit state with success
```

```
                          004D        .SBTTL  FAL$FIL_PARSE
                          004D
                          004D ;++
                          004D ; This routine parses the file specification received from partner for file
                          004D ; access functions other than the DAP DIRECTORY list function.
                          004D ;
                          004D ; Note: If the $PARSE operation finds a wildcard character, but the partner
                          004D ;       process declares that it does not support wildcard operations, then
                          004D ;       an RMS$_WLD error will be returned. This is done to assist systems
                          004D ;       (such as DECnet-IAS) that do not understand VAX/VMS wildcard file
                          004D ;       specifications, and therefore cannot determine that one has been
                          004D ;       specified.
                          004D ;--
                          004D
                          004D FAL$FIL_PARSE::                                    ; Entry point
                          004D        $PARSE  FAB=R10                             ; Parse the file specification
                0AF6  30  0056        BSBW    CHECK_FOR_PMR                       ; Possibly disallow poor-man's routing
               19 50  E9  0059        BLBC    R0,20$                              ; Branch on failure
                  08  E1  005C        BBC     #NAM$V_WILDCARD,-                   ; Branch if non-wild expanded string
          10 02C8 C8      005E                FAL$L_NAM+NAM$L_FNB(R8),10$
                          0062        $SETBIT #FAL$V_WILD,(R8)                    ; Denote wildcard operation
                  26  E0  0066        BBS     #DAP$V_WILDCARD,-                   ; Branch if partner supports wildcard
             07 28 A9     0068                DAP$Q_SYSCAP(R9),10$                ;  operations
     50     8744 8F   3C  006B        MOVZWL  #<RMS$_WLD&^XFFFF>,R0               ; If not, return an error code for
                  03  11  0070        BRB     20$                                 ;  'invalid wildcard operation'
                0C71  31  0072 10$:   BRW     EXIT_SUCCESS                        ; Exit state with success
                0C16  31  0075 20$:   BRW     ERR_FILE_OPEN                       ; Return error in Status message
                          0078
                          0078
                          0078        .SBTTL  FAL$CHECK_WILD
                          0078
                          0078 ;++
                          0078 ; This routine determines whether or not the parsed file specification contains
                          0078 ; any wildcard characters. The wildcard status bit stored in the FAL work area
                          0078 ; is used to form the state transition status code.
                          0078 ;--
                          0078
                          0078 FAL$CHECK_WILD::                                   ; Entry point
   50   68   01   0A  EF  0078        EXTZV   #FAL$V_WILD,#1,(R8),R0              ; Form yes/no reply
                      05  007D        RSB                                         ; Exit state with status code in R0
```

```
                     007E          .SBTTL  FAL$FIL_SEARCH
                     007E
                     007E  ;++
                     007E  ; This routine performs the search operation on a wildcard file specification
                     007E  ; for file access functions other than the DAP DIRECTORY list function.
                     007E  ;--
                     007E
                     007E  FAL$FIL_SEARCH::                            ; Entry point
                     007E          $SEARCH FAB=R10                     ; Search for next file name match
       12 50     E9  0087          BLBC    R0,10$                      ; Branch on failure
                     008A
                     008A  ;
                     008A  ; Re-establish the FHCXAB at the head of the XAB chain.
                     008A  ;
                     008A
     02F4 C8     DE  008A          MOVAL   FAL$L_FHCXAB(R8),-          ; Store FHCXAB pointer in XAB chain
       24 AA         008E                  FAB$L_XAB(R10)              ;
     02F8 C8     DE  0090          MOVAL   FAL$L_FHCXAB+XAB$L_NXT(R8),-
       7C A8         0094                  FAL$L_CHAIN_NXT(R8)         ; Save address of next chain pointer
       7C B8     D4  0096          CLRL    @FAL$L_CHAIN_NXT(R8)        ; Terminate XAB chain at FHCXAB
         0C4A    31  0099          BRW     EXIT_SUCCESS                ; Exit state with success
                     009C
                     009C  ;
                     009C  ; Note that RMS$_NMF converts to an Access Complete message.
                     009C  ;
                     009C
 82CA 8F    50   B1  009C  10$:    CMPW    R0,#<RMS$_NMF&^XFFFF>       ; Check for normal termination
            03   13  00A1          BEQL    20$                         ;  of wildcard operation
          0BE8   31  00A3          BRW     ERR_FILE_OPEN               ; Return error in Status message
          0AC6   30  00A6  20$:    BSBW    SEND_CMP                    ; Send Access Complete message
          0C3A   31  00A9          BRW     EXIT_SUCCESS                ; Exit state with success
                     00AC
                     00AC
                     00AC          .SBTTL  FAL$CHECK_NMF
                     00AC
                     00AC  ;++
                     00AC  ; This routine determines whether or not an RMS$_NMF (no more files found)
                     00AC  ; completion code was returned on the last $SEARCH call.
                     00AC  ;--
                     00AC
                     00AC  FAL$CHECK_NMF::                             ; Entry point
       08 AA     B1  00AC          CMPW    FAB$L_STS(R10),-            ; Branch if last error code was
     82CA 8F         00AF                  #<RMS$_NMF&^XFFFF>          ;  RMS$_NMF
            03   13  00B2          BEQL    10$                         ;
          0C1F   31  00B4          BRW     EXIT_FAILURE                ; Exit state with failure
          0C2C   31  00B7  10$:    BRW     EXIT_SUCCESS                ; Exit state with success
```

```
                      00BA        .SBTTL  FAL$OPEN
                      00BA
                      00BA  ;++
                      00BA  ; This routine performs the DAP OPEN file function for both single file and
                      00BA  ; wildcard requests.
                      00BA  ;--
                      00BA
                      00BA  FAL$OPEN::                                     ; Entry point
                      00BA
                      00BA  ;
                      00BA  ; An Attributes message must precede the Access message. Check for it.
                      00BA  ;
                      00BA
   03 68    01    E0  00BA         BBS     #FAL$V_ATT_MSG,(R8),10$ ; Branch if ok
          0C04    31  00BE         BRW     ERR_SYNCHRONIZE         ; Return error in Status message
                      00C1
                      00C1  ;
                      00C1  ; On a wildcard operation return Name messages derived from the resultant name
                      00C1  ; string obtained on the $SEARCH call.
                      00C1  ;
                      00C1
   03 68    0A    E1  00C1  10$:   BBC     #FAL$V_WILD,(R8),20$    ; Branch if not wildcard operation
          0B4E    30  00C5         BSBW    SEND_3PART_NAM          ; Send resultant name in three Name
                      00C8                                         ;  message format
                      00C8
                      00C8  ;
                      00C8  ; Open the specified file.
                      00C8  ;
                      00C8
          0975    30  00C8  20$:   BSBW    CHAIN_RQST_XABS         ; Add required XABs to XAB chain
         14 AA    B4  00CB         CLRW    FAB$W_DEQ(R10)          ; Clear the old DEQ value
                      00CE         $OPEN   FAB=R10                 ; Open the file
         36 50    E9  00D7         BLBC    R0,60$                  ; Branch on failure
         FF23'    30  00DA         BSBW    FAL$LOG_RESNAM          ; Log resultant name in print file
   06 69    26    E0  00DD         BBS     #DAP$V_GEQ_V70,(R9),30$ ; Branch if partner uses DAP since V7.0
         1F AA    91  00E1         CMPB    FAB$B_RFM(R10),-        ; Return error if file has a record
            04        00E4                 #FAB$C_STM              ;  format that cannot be mapped into
            21    1A  00E5         BGTRU   50$                     ;  DAP RFM field of an earlier DAP spec
                      00E7
                      00E7  ;
                      00E7  ; Set-up to compute file level CRC checksum if requested.
                      00E7  ;
                      00E7
            03    E1  00E7  30$:   BBC     #DAP$V_RET_CRC,-        ; Branch if checksum option not
      08 01F5     C8  00E9                 FAL$B_ACCOPT(R8),40$    ;  requested by accessing node
     0000FFFF 8F  D0  00ED         MOVL    #DAP$K_CRC_INIT,-       ; Use initial CRC value as first
         20 A9        00F3                 DAP$L_CRC_RSLT(R9)      ;  CRC resultant value
                      00F5
                      00F5  ;
                      00F5  ; Return (main) Attributes, Extended Attributes, and (resultant) Name messages
                      00F5  ; to partner as directed by request mask, terminated by an Acknowledge message.
                      00F5  ;
                      00F5
          097F    30  00F5  40$:   BSBW    CHAIN_RQST_PRT2         ; Fill in alternate ALLXABs and KEYXABs
                      00F8                                         ;  as directed by NOA and NOK values
         15 50    E9  00F8         BLBC    R0,60$                  ; Branch on failure
          0A85    30  00FB         BSBW    SEND_OPTNL_MSGS         ; Build and send optional DAP messages
```

```
                        00FE   486         $SETBIT #FAL$V_LAST_MSG,(R8)      ; Declare this last message to block
              0A5D  30  0102   487         BSBW    SEND_ACK                  ; Send Acknowledge message
              0BDE  31  0105   488         BRW     EXIT_SUCCESS              ; Exit state with success
    50   8664 8F   3C  0108   489 50$:    MOVZWL  #<RMS$_RFM&^XFFFF>,R0     ; Generate invalid record format error
              0C AA  D4  010D   490         CLRL    FAB$L_STV(R10)            ; Zero secondary status value
              0B7B  31  0110   491 60$:    BRW     ERR_FILE_OPEN             ; Return error in Status message
```

```
                              0113          .SBTTL  FAL$SUBMIT, FAL$CREATE
                              0113
                              0113  ;++
                              0113  ; The following routines have separate entry points but share common code ...
                              0113  ;
                              0113  ; This routine performs the DAP SUBMIT file function which consists of creating
                              0113  ; the command file, submitting it to the symbiont manager for execution, and
                              0113  ; finally deleting the command file when the job is finished.
                              0113  ;--
                              0113
                              0113  FAL$SUBMIT::                                    ; Entry point
 04 AA   0000C000 8F   C8     0113          BISL2   #<<FAB$M_SCF>!-                 ; Set submit-on-close and
                              011B                    <FAB$M_DLT>!-                 ;  delete-after-execution bits
                              011B                  0>,FAB$L_FOP(R10)               ; Fall thru to FAL$CREATE ...
                              011B
                              011B  ;++
                              011B  ; This routine performs the DAP CREATE file function.
                              011B  ;--
                              011B
                              011B  FAL$CREATE::                                    ; Entry point
                              011B
                              011B  ;
                              011B  ; An Attributes message must precede the Access message. Check for it.
                              011B  ;
                              011B
       03 68   01      E0     011B          BBS     #FAL$V_ATT_MSG,(R8),10$ ; Branch if ok
          0BA3         31     011F          BRW     ERR_SYNCHRONIZE         ; Return error in Status message
                              0122
                              0122          ASSUME  FAB$C_SEQ EQ 0
        1D AA          95     0122  10$:    TSTB    FAB$B_ORG(R10)          ; Is this a sequential file?
           12          12     0125          BNEQ    15$                     ; Branch if it isn't.
           01          91     0127          CMPB    #FAB$C_FIX,-            ; Is this a fixed record format?
        1F AA                 0129                  FAB$B_RFM(R10)
           08          12     012B          BNEQ    12$                     ; Branch if not
  36 AA    0200 8F     B1     012D          CMPW    #512,FAB$W_MRS(R10)     ; Is the recordsize right for sharing?
           04          13     0133          BEQL    15$                     ; Branch if it is
           02          8A     0135  12$:    BICB2   #FAB$M_SHRGET,-         ; Only allow sharing to be set on files
        17 AA                 0137                  FAB$B_SHR(R10)          ; that can be.  Some systems will try
                              0139                                          ; to always set SHRGET regardless.
                              0139
                              0139  ;
                              0139  ; Create the specified file.
                              0139  ;
                              0139
     0F 68   33        E1     0139  15$:    BBC     #FAL$V_DIS_PMR,(R8),20$ ; If poor-man's routing is not allowed
                              013D          $PARSE  FAB=R10                 ;  then first parse the filespec to
          0A06         30     0146          BSBW    CHECK_FOR_PMR           ;  determine if a node name is present
         2D 50         E9     0149          BLBC    R0,40$                  ; Branch on failure (node name found)
          0984         30     014C  20$:    BSBW    CHAIN_RECV_XABS         ; Add received XABs to XAB chain
                              014F          $CREATE FAB=R10                 ; Create the file
         1E 50         E9     0158          BLBC    R0,40$                  ; Branch on failure
         FEA2'         30     015B          BSBW    FAL$LOG_RESNAM          ; Log resultant name in print file
                              015E
                              015E  ;
                              015E  ; Set-up to compute file level CRC checksum if requested.
                              015E  ;
                              015E
```

```
           03  E1  015E   550          BBC     #DAP$V_RET_CRC,-          ; Branch if checksum option not
   08 01F5 C8      0160   551                  FAL$B_ACCOPT(R8),30$      ;  requested by accessing node
 0000FFFF 8F  D0  0164   552          MOVL    #DAP$K_CRC_INIT,-          ; Use initial CRC value as first
        20 A9      016A   553                  DAP$L_CRC_RSLT(R9)        ;  CRC resultant value
                   016C   554
                   016C   555 ;
                   016C   556 ; Return (main) Attributes, Extended Attributes, and (resultant) Name messages
                   016C   557 ; to partner as directed by request mask, followed by an Acknowledge message.
                   016C   558 ;
                   016C   559
        0A14   30  016C   560 30$:     BSBW    SEND_OPTNL_MSGS           ; Build and send optional DAP messages
                   016F   561          $SETBIT #FAL$V_LAST_MSG,(R8)      ; Declare this last message to block
        09EC   30  0173   562          BSBW    SEND_ACK                  ; Send Acknowledge message
        0B6D   31  0176   563          BRW     EXIT_SUCCESS              ; Exit state with success
        0B12   31  0179   564 40$:     BRW     ERR_FILE_OPEN             ; Return error in Status message
```

```
                   017C         .SBTTL  FAL$ERASE
                   017C
                   017C  ;++
                   017C  ; This routine performs the DAP ERASE file function.
                   017C  ;
                   017C  ; Note: RMS does not return XAB information on $ERASE. The DAP specification,
                   017C  ;       however, allows partner to request that DAP Extended Attributes messages
                   017C  ;       be returned, but it is unlikely that anyone will make a request for
                   017C  ;       anything but a Name message.
                   017C  ;--
                   017C
                   017C  FAL$ERASE::                                        ; Entry point
      08C1    30   017C          BSBW    CHAIN_RQST_XABS                    ; Add required XABs to XAB chain
                   017F          $ERASE  FAB=RTO                            ; Erase (delete) the file
     1C 50    E9   0188          BLBC    R0,30$                             ; Branch on failure
     FE72'    30   018B          BSBW    FAL$LOG_RESNAM                     ; Log resultant name in print file
     70 A8    B5   018E          TSTW    FAL$W_DISPLAY(R8)                  ; Branch if no additional information
        0A    13   0191          BEQL    10$                                ;  to return
      09ED    30   0193          BSBW    SEND_OPTNL_MSGS                    ; Build and send optional DAP messages
  03 69 26    E1   0196          BBC     #DAP$V_GEQ_V70,(R9),10$            ; Send Acknowledge message only if
      09C5    30   019A          BSBW    SEND_ACK                           ;  partner uses DAP since V7.0
  03 68 0A    E0   019D  10$:    BBS     #FAL$V_WILD,(R8),20$               ; Branch if wildcard operation
      09CB    30   01A1          BSBW    SEND_CMP                           ; Send Access Complete message
      0B3F    31   01A4  20$:    BRW     EXIT_SUCCESS                       ; Exit state with success
  03 68 0A    E1   01A7  30$:    BBC     #FAL$V_WILD,(R8),40$               ; Branch if not wildcard operation
      0A68    30   01AB          BSBW    SEND_3PART_NAM                     ; Send resultant name in three Name
                   01AE                                                     ;  message format
      0ADD    31   01AE  40$:    BRW     ERR_FILE_OPEN                      ; Return error in Status message
```

```
                           01B1          .SBTTL  FAL$RENAME
                           01B1
                           01B1   ;++
                           01B1   ; This routine performs the DAP RENAME file function.
                           01B1   ;--
                           01B1
                           01B1   FAL$RENAME::                                   ; Entry point
          03 69    26  E0  01B1          BBS     #DAP$V_GEQ_V70,(R9),10$       ; Exit if partner does not support
             0870      31  01B5          BRW     FAL$UNS_ACCFUNC               ;  DAP V7.0
                           01B8
                           01B8   ;
                           01B8   ; Set up registers
                           01B8   ;
                           01B8
             0C00 8F   BB  01B8   10$:   PUSHR   #^M<R10,R11>                  ; Save registers
               56  5A  D0  01BC          MOVL    R10,R6                        ; Get old filename FAB address
            57  28 A6  D0  01BF          MOVL    FAB$L_NAM(R6),R7              ; Get old filename NAM address
          5A  0800 C8  DE  01C3          MOVAL   FAL$L_FAB2(R8),R10            ; Get new filename FAB address (FAB2)
            5B  28 AA  D0  01C8          MOVL    FAB$L_NAM(R10),R11            ; Get new filename NAM address (NAM2)
   5E    00000050 8F   C2  01CC          SUBL2   #FAL$K_FAB,SP                 ; Make room on stack for temporary
               5C  5E  D0  01D3          MOVL    SP,AP                         ;  FAB (old filename) and save pointer
                           01D6
                           01D6   ;
                           01D6   ; Create a temporary FAB (old filename) on the stack and fill in or clear
                           01D6   ; appropriate fields.
                           01D6   ;
                           01D6
   6E    66   0050 8F  28  01D6          MOVC3   #FAL$K_FAB,(R6),(SP)          ; Copy old FAB onto the stack
          10 68    0A  E1  01DC          BBC     #FAL$V_WILD,(R8),20$          ; Branch in non-wild operation
               04 A7   D0  01E0          MOVL    NAM$L_RSA(R7),-               ; Since an open by NAM block is
               2C AE       01E3                  FAB$L_FNA(SP)                 ;  not allowed, overwrite the FNA
               03 A7   90  01E5          MOVB    NAM$B_RSL(R7),-               ;  and FNS with the RSA and RSL
               34 AE       01E8                  FAB$B_FNS(SP)                 ;
               02 AE   B4  01EA          CLRW    FAB$W_IFI(SP)                 ; Clear the IFI field
               28 AE   D4  01ED          CLRL    FAB$L_NAM(SP)                 ; Clear the NAM field (so that the
                           01F0                                                ;  wildcard context is not touched
                           01F0                                                ;  by the $RENAME)
                           01F0
                           01F0   ;
                           01F0   ; Parse the new filename against the old filename to resolve any wildcards
                           01F0   ; or empty fields - i.e. rename a.a to .b or a.a to *.b
                           01F0   ;
                           01F0
               2C AA   DD  01F0   20$:   PUSHL   FAB$L_FNA(R10)                ; Save the new FNA on the stack
            7E  34 AA  9A  01F3          MOVZBL  FAB$B_FNS(R10),-(SP)          ; Save the new FNS on the stack
            10 AB  57  D0  01F7          MOVL    R7,NAM$L_RLF(R11)             ; Use old filename NAM as RLF
                           01FB          $SETBIT #FAB$V_OFP,FAB$L_FOP(R10);Set the output file parse bit
                 FE4A  30  0200          BSBW    FAL$FIL_PARSE                 ; Parse the new filename
               0C AB   D0  0203          MOVL    NAM$L_ESA(R11),-              ; Overwrite the FNA with the old
               2C AA       0206                  FAB$L_FNA(R10)                ;  filename ESA and ESL
               0B AB   90  0208          MOVB    NAM$B_ESL(R11),-              ;
               34 AA       020B                  FAB$B_FNS(R10)                ;
                           020D
                           020D   ;
                           020D   ; Any wildcards have now been resolved. Do the $RENAME, restore the registers
                           020D   ; and stack, process optional messages, do logging and exit.
                           020D   ;
```

```
                           020D
                           020D          $RENAME OLDFAB=AP,NEWFAB=R10      ; Rename the file
                  02   BA  021C          POPR    #^M<R1>                   ; Get the new FNS
          34 AA   51   90  021E          MOVB    R1,FAB$B_FNS(R10)         ; Restore the new FNS
             2C AA   8ED0  0222          POPL    FAB$L_FNA(R10)            ; Restore the new FNA
  5E     00000050 8F   C0  0226          ADDL2   #FAL$K_FAB,SP             ; Evaporate temporary old FAB
             0C00 8F   BA  022D          POPR    #^M<R10,R11>              ; Restore registers (old FAB in R10)
              2A 50    E9  0231          BLBC    R0,40$                    ; Branch if $RENAME failed
                FDC9'  30  0234          BSBW    FAL$LOG_RESNAM            ; Log old resultant name in print file
                0949   30  0237          BSBW    SEND_OPTNL_MSGS           ; Build and send old optional DAP msgs
                0925   30  023A          BSBW    SEND_ACK                  ; Send Acknowledge message
        5A   0800 C8   DE  023D          MOVAL   FAL$C_FAB2(R8),R10        ; Set up with new FAB
                           0242          $SETBIT #FAL$V_NEWNAM,(R8)        ; Set flag to send new resultant nam msg
                FDB7'  30  0246          BSBW    FAL$LOG_RESNAM            ; Log new resultant name in print file
                0937   30  0249          BSBW    SEND_OPTNL_MSGS           ; Build and send new optional DAP msgs
                0913   30  024C          BSBW    SEND_ACK                  ; Send Acknowledge message
        5A   0200 C8   DE  024F          MOVAL   FAL$C_FAB(R8),R10         ; Restore old FAB address
          03 68   0A   E0  0254          BBS     #FAL$V_WILD,(R8),30$      ; Branch if wildcard operation
                0914   30  0258          BSBW    SEND_CMP                  ; Send Access Complete message
                0A88   31  025B  30$:    BRW     EXIT_SUCCESS              ; Exit state with success
          03 68   0A   E1  025E  40$:    BBC     #FAL$V_WILD,(R8),50$      ; Branch if not wildcard operation
                09B1   30  0262          BSBW    SEND_3PART_NAM            ; Send resultant name in three Name
                           0265                                            ;  message format
                0A26   31  0265  50$:    BRW     ERR_FILE_OPEN             ; Return error in Status message
```

```
                     0268        .SBTTL  FAL$EXECUTE
                     0268
                     0268  ;++
                     0268  ; This routine performs the DAP EXECUTE file function which consists of
                     0268  ; submitting a command file to the symbiont manager for execution. The command
                     0268  ; file must exist on the destination node and it is not deleted after execution.
                     0268  ;
                     0268  ; Note: The DAP specification allows partner to request that DAP Extended
                     0268  ;       Attributes messages be returned, but it is unlikely that anyone will
                     0268  ;       make a request for anything but a Name message.
                     0268  ;--
                     0268
                     0268  FAL$EXECUTE::                                        ; Entry point
                     0268          $SETBIT #FAB$V_SCF,FAB$L_FOP(R10);Set submit-on-close bit
        07D0   30    026D          BSBW    CHAIN_RQST_XABS               ; Add required XABs to XAB chain
                     0270          $OPEN   FAB=R10                       ; Open the file
     32 50     E9    0279          BLBC    R0,30$                        ; Branch on failure
       FD81'   30    027C          BSBW    FAL$LOG_RESNAM                ; Log resultant name in print file
     24 AA     D4    027F          CLRL    FAB$L_XAB(R10)                ; Remove any XABs from chain
                     0282          $CLOSE  FAB=R10                       ; Close the file and submit it
     2A 50     E9    028B          BLBC    R0,50$                        ; Branch on failure
51   04 AA     D0    028E          MOVL    FAB$L_FOP(R10),R1             ; Get FOP options
       FD6B'   30    0292          BSBW    FAL$LOG_CLSMSG                ; Log file close in print file
     70 A8     B5    0295          TSTW    FAL$W_DISPLAY(R8)             ; Branch if no additional information
        0A     13    0298          BEQL    10$                           ;  to return
      08E6     30    029A          BSBW    SEND_OPTNL_MSGS               ; Build and send optional DAP messages
03 69   26     E1    029D          BBC     #DAP$V_GEQ_V70,(R9),10$       ; Send Acknowledge message only if
      08BE     30    02A1          BSBW    SEND_ACK                      ;  partner uses DAP since V7.0
03 68   0A     E0    02A4  10$:    BBS     #FAL$V_WILD,(R8),20$          ; Branch if wildcard operation
      08C4     30    02A8          BSBW    SEND_CMP                      ; Send Access Complete message
      0A38     31    02AB  20$:    BRW     EXIT_SUCCESS                  ; Exit state with success
03 68   0A     E1    02AE  30$:    BBC     #FAL$V_WILD,(R8),40$          ; Branch if not wildcard operation
      0961     30    02B2          BSBW    SEND_3PART_NAM                ; Send resultant name in three Name
                     02B5                                                ;  message format
      09D6     31    02B5  40$:    BRW     ERR_FILE_OPEN                 ; Return error in Status message
03 68   0A     E1    02B8  50$:    BBC     #FAL$V_WILD,(R8),60$          ; Branch if not wildcard operation
      0957     30    02BC          BSBW    SEND_3PART_NAM                ; Send resultant name in three Name
                     02BF                                                ;  message format
      09DE     31    02BF  60$:    BRW     ERR_FILE_CLOS                 ; Return error in Status message
```

```
                      02C2           .SBTTL  FAL$DIR_PARSE
                      02C2
                      02C2   ;++
                      02C2   ; This routine performs the parse phase of the DAP DIRECTORY list function.
                      02C2   ;--
                      02C2
                      02C2   FAL$DIR_PARSE::                              ; Entry point
         0000'CF  90  02C2           MOVB    W^FAL$GQ_WILDSPEC,-          ; Store size of wildcard string used
           35 AA      02C6                   FAB$B_DNS(R10)               ;  as default file name (namely *.*;*)
         0004'CF  D0  02C8           MOVL    W^FAL$GQ_WILDSPEC+4,-        ; Store address of default wildcard file
           30 AA      02CC                   FAB$L_DNA(R10)               ;  specification string buffer
                      02CE           $PARSE  FAB=R10                      ; Parse the directory file specification
            0875  30  02D7           BSBW    CHECK_FOR_PMR                ; Possibly disallow poor-man's routing
           18 50  E9  02DA           BLBC    R0,20$                       ; Branch on failure
           35 AA  94  02DD           CLRB    FAB$B_DNS(R10)               ; Discard default file name string
                      02E0                                                ;  in case an $OPEN is performed
            075D  30  02E0           BSBW    CHAIN_RQST_XABS              ; Add required XABs to XAB chain for
                      02E3                                                ;  use on subsequent $SEARCH or $OPEN
   17 AA   43 8F  90  02E3           MOVB    #<<FAB$M_SHRGET>!-           ; Specify shared access to files
                      02E8                     <FAB$M_SHRPUT>!-           ;  so that subsequent $OPEN to obtain
                      02E8                     <FAB$M_UPI>!-              ;  file attributes will be less likely
                      02E8                   0>,FAB$B_SHR(R10)            ;  to fail with RMS$_FLK error
              08  E1  02E8           BBC     #NAM$V_WILDCARD,-            ; Branch if non-wild expanded string
      04 02C8 C8      02EA                   FAL$L_NAM+NAM$L_FNB(R8),10$
                      02EE           $SETBIT #FAL$V_WILD,(R8)             ; Denote wildcard operation
            09F1  31  02F2   10$:    BRW     EXIT_SUCCESS                 ; Exit state with success
            0996  31  02F5   20$:    BRW     ERR_FILE_OPEN                ; Return error in Status message
                      02F8
                      02F8
                      02F8           .SBTTL  FAL$DIR_END
                      02F8
                      02F8   ;++
                      02F8   ; This routine performs the final phase of the DAP DIRECTORY list function.
                      02F8   ;--
                      02F8
                      02F8   FAL$DIR_END::                                ; Entry point
            0874  30  02F8           BSBW    SEND_CMP                     ; Send Access Complete message
            09E8  31  02FB           BRW     EXIT_SUCCESS                 ; Exit state with success
```

```
                      02FE          .SBTTL  FAL$DIR_SEARCH
                      02FE
                      02FE  ;++
                      02FE  ; This routine performs the search phase of the DAP DIRECTORY list function.
                      02FE  ;
                      02FE  ; Note: This routine performs only one search operation and therefore it
                      02FE  ;       must be called repeatedly to find all requested files.
                      02FE  ;--
                      02FE
                      02FE  FAL$DIR_SEARCH::                                 ; Entry point
   57    28 AA    D0  02FE          MOVL    FAB$L_NAM(R10),R7                ; Get address of NAM block
      0100 8F     AB  0302          BICW3   #DAP$M_DSP_NAM,-                 ; Are any attributes requested
   50    70 A8        0306                  FAL$W_DISPLAY(R8),R0             ;  (excluding resultant name)?
         0A       13  0309          BEQL    10$                              ; Branch if not
         11       E1  030B          BBC     #NAM$V_NODE,-                    ; Branch if this is the end node
      05 34 A7        030D                  NAM$L_FNB(R7),10$                ;  (i.e., not an intermediate node)
                      0310          $SETBIT #NAM$V_SRCHXABS,NAM$B_NOP(R7)
                      0315                                                   ; Request return of XAB info to avoid
                      0315                                                   ;  opening file to get its attributes
                      0315  10$:    $SEARCH FAB=R10                          ; Search for next file specification
      08 50       E9  031E          BLBC    R0,20$                           ; Branch on failure
      FCDC'       30  0321          BSBW    FAL$LOG_RESNAM                   ; Log resultant name in print file
                      0324
                      0324  ;
                      0324  ; Send (three part) Name messages to partner.
                      0324  ;
                      0324
      08EF        30  0324          BSBW    SEND_3PART_NAM                   ; Send resultant name in three Name
                      0327                                                   ;  message format
         10       11  0327          BRB     SEND_ATTRIBUTES                  ; Send requested attribute information
                      0329
                      0329  ;
                      0329  ; Note that RMS$_NMF is converted to an Access Complete message.
                      0329  ;
                      0329
 82CA 8F     50   B1  0329  20$:    CMPW    R0,#<RMS$_NMF&^XFFFF>            ; Check for normal termination
         03       13  032E          BEQL    30$                              ;  of directory search sequence
      095B        31  0330          BRW     ERR_FILE_OPEN                    ; Return error in Status message
      0839        30  0333  30$:    BSBW    SEND_CMP                         ; Send Access Complete message
      09AD        31  0336          BRW     EXIT_SUCCESS                     ; Exit state with success
                      0339
                      0339  ;+
                      0339  ; Open and close the file specified in the resultant string to obtain its
                      0339  ; attributes, unless this is an intermediate node. (If the file name given to
                      0339  ; $SEARCH contained a node name, then attributes have already been returned.)
                      0339  ;-
                      0339
                      0339  SEND_ATTRIBUTES:                                 ;
   57    28 AA    D0  0339          MOVL    FAB$L_NAM(R10),R7                ; Get address of NAM block
      0100 8F     AB  033D          BICW3   #DAP$M_DSP_NAM,-                 ; Are any attributes requested
   50    70 A8        0341                  FAL$W_DISPLAY(R8),R0             ;  (excluding resultant name)?
         37       13  0344          BEQL    30$                              ; Branch if not
         11       E1  0346          BBC     #NAM$V_NODE,-                    ; Branch if this is the end node
   0A    34 A7        0348                  NAM$L_FNB(R7),10$                ;  (i.e., not an intermediate node)
   51    0C AA    D0  034B          MOVL    FAB$L_STV(R10),R1                ; Branch if not partial success
         29       13  034F          BEQL    20$                              ;  (i.e., file attributes were returned
                      0351                                                   ;  as requested on $SEARCH)
```

```
           46   10 0351  813             BSBB    MAP_SS_TO_RMS                ; Convert SS code to RMS code and
           3E   11 0353  814             BRB     60$                          ;  say error occurred on file open
                   0355  815
                   0355  816 ;
                   0355  817 ; Now open and close the file to obtain file attributes. Note that the required
                   0355  818 ; XABs have already been chained into the XAB chain by FAL$DIR_PARSE.
                   0355  819 ;
                   0355  820
        14 AA   B4 0355  821 10$:        CLRW    FAB$W_DEQ(R10)               ; Clear the old DEQ value
                   0358  822             $OPEN   FAB=R10                      ; Open the file
        23 50   E9 0361  823             BLBC    R0,50$                       ; Branch on failure
        24 AA   DD 0364  824             PUSHL   FAB$L_XAB(R10)               ; Save XAB chain pointer
        24 AA   D4 0367  825             CLRL    FAB$L_XAB(R10)               ; Remove any XABs from chain
                   036A  826             $CLOSE  FAB=R10                      ; Close the file
        24 AA 8ED0 0373  827             POPL    FAB$L_XAB(R10)               ; Restore XAB chain pointer
        1C 50   E9 0377  828             BLBC    R0,70$                       ; Branch on failure
                   037A  829
                   037A  830 ;
                   037A  831 ; Return (main) Attributes, Extended Attributes, and (resultant) Name messages
                   037A  832 ; to partner as directed by request mask.
                   037A  833 ;
                   037A  834
         0806   30 037A  835 20$:        BSBW    SEND_OPTNL_MSGS              ; Build and send optional DAP messages
                   037D  836
                   037D  837 ;
                   037D  838 ; Send Acknowledge message to partner and exit state successfully.
                   037D  839 ;
                   037D  840
   03 69   26   E1 037D  841 30$:        BBC     #DAP$V_GEQ_V70,(R9),40$ ; Send Acknowledge message only if
         07DE   30 0381  842             BSBW    SEND_ACK                     ;  partner uses DAP since V7.0
         095F   31 0384  843 40$:        BRW     EXIT_SUCCESS                 ; Exit state with success
                   0387  844
                   0387  845 ;
                   0387  846 ; Error processing.
                   0387  847 ;
                   0387  848
 8292 8F   50   B1 0387  849 50$:        CMPW    R0,#<RMS$_FNF&^XFFFF>        ; Convert RMS$_FNF to RMS$_ATR if
           05   12 038C  850             BNEQ    60$                          ;  $SEARCH succeeded but subsequent
 50   COCC 8F   3C 038E  851             MOVZWL  #<RMS$_ATR&^XFFFF>,R0        ;  $OPEN failed because file does not
                   0393  852                                                  ;  exist although directory entry does
         08F8   31 0393  853 60$:        BRW     ERR_FILE_OPEN                ; Return error in Status message
         0907   31 0396  854 70$:        BRW     ERR_FILE_CLOS                ; Return error in Status message
                   0399  855
                   0399  856 ;+
                   0399  857 ; This routine maps a System Service code to an RMS completion code.
                   0399  858 ;-
                   0399  859
                   0399  860 MAP_SS_TO_RMS:                                 ; Entry point
 0800 8F   51   B1 0399  861             CMPW    R1,#SS$_ACCONFLICT           ; Check for file access locked error
           06   12 039E  862             BNEQ    10$                          ; Branch if no match
 50   828A 8F   3C 03A0  863             MOVZWL  #<RMS$_FLK&^XFFFF>,R0        ; Convert to FLK (lower 16 bits)
           05      03A5  864             RSB                                  ; Exit
      24   51   B1 03A6  865 10$:        CMPW    R1,#SS$_NOPRIV               ; Check for privilege violation
           06   12 03A9  866             BNEQ    20$                          ; Branch if no match
 50   829A 8F   3C 03AB  867             MOVZWL  #<RMS$_PRV&^XFFFF>,R0        ; Convert to PRV (lower 16 bits)
           05      03B0  868             RSB                                  ; Exit
 08A8 8F   51   B1 03B1  869 20$:        CMPW    R1,#SS$_FILELOCKED           ; Check for file deaccess locked error
```

```
             06   12  03B6   870        BNEQ    30$                         ; Branch if no match
 50    C002 8F    3C  03B8   871        MOVZWL  #<RMS$_ACC&^XFFFF>,R0       ; Convert to ACC (lower 16 bits)
                  05  03BD   872        RSB                                 ; Exit
0888 8F      51   B1  03BE   873 30$:   CMPW    R1,#SS$_FCPREADERR          ; Check for read attributes error
             00   12  03C3   874        BNEQ    40$                         ; Branch if no match; map to ATR
 50    COCC 8F    3C  03C5   875 40$:   MOVZWL  #<RMS$_ATR&^XFFFF>,R0       ; Convert to ATR (lower 16 bits)
                  05  03CA   876        RSB                                 ; Exit
```

```
                          03CB          .SBTTL  FAL$LOAD_IMAGE
                          03CB
                          03CB  ;++
                          03CB  ; This routine performs the VMS system specific load image function to support
                          03CB  ; the DCL command '$RUN node::file.exe'. To accomplish this, FAL ceases to use
                          03CB  ; DAP after the file is opened to send the image file to the VMS image activator
                          03CB  ; at the requesting node. Instead, FAL sends the image file in 512 byte chunks
                          03CB  ; via QIOs until end-of-file is reached, then waits for the requesting node to
                          03CB  ; disconnect the logical link.
                          03CB  ;--
                          03CB
                          03CB  FAL$LOAD_IMAGE::                                      ; Entry point
                          03CB          $CONNECT RAB=R11                              ; Establish a record stream
            44 50   E9    03D4          BLBC    R0,30$                                ; Branch on failure
      51    4C A8   D0    03D7          MOVL    FAL$Q_XMT+4(R8),R1                    ; Get address of transmit buffer
   54 A8       51   D0    03DB          MOVL    R1,FAL[$Q_BLD+4(R8)                   ; Initialize address in BLD descriptor
   24 AB       51   D0    03DF          MOVL    R1,RAB$L_UBF(R11)                     ; Initialize buffer address in RAB
            1A A8   B0    03E3          MOVW    FAL$W_DAPBUFSIZ(R8),-                 ; Put size of transmit buffer
            20 AB         03E6                  RAB$W_USZ(R11)                        ;  in RAB
   57    00C0 C8    DE    03E8          MOVAL   FAL$L_STB(R8),R7                      ; Get address of Statistics block
                          03ED  10$:    $GET    RAB=R11                               ; Get the next record
            18 50   E9    03F6          BLBC    R0,20$                                ; Branch on error
      54    22 AB   3C    03F9          MOVZWL  RAB$W_RSZ(R11),R4                     ; Update BLD descriptor with size
   50 A8       54   D0    03FD          MOVL    R4,FAL[$Q_BLD(R8)                     ;  of record (512 bytes for image files)
                          0401          $SETBIT #FAL$V_LAST_MSG,(R8)                  ; Declare this last message to block
           FBF8'    30    0405          BSBW    FAL$TRANSMIT                          ; Send record to partner
            1C A7   D6    0408          INCL    FAL$L_XMT_DAT(R7)                     ; Increment XMT record/block count
   20 A7       54   C0    040B          ADDL2   R4,FAL[$L_XMT_USR(R7)                 ; Update XMT user data byte count
               DC   11    040F          BRB     10$                                   ; Loop until EOF or error
 827A 8F       50   B1    0411  20$:    CMPW    R0,#<RMS$_EOF&^XFFFF>                 ; Is it an end-of-file?
               03   12    0416          BNEQ    30$                                   ; Branch if not
             08CB   31    0418          BRW     EXIT_SUCCESS                          ; Exit state with success
             08B8   31    041B  30$:    BRW     EXIT_FAILURE                          ; Exit state with failure
```

```
                         041E          .SBTTL  FAL$CONNECT
                         041E
                         041E   ;++
                         041E   ; This routine performs the DAP CONNECT record stream function.
                         041E   ;--
                         041E
                         041E   FAL$CONNECT::                                  ; Entry point
             66 A8   B4  041E          CLRW    FAL$Q_RMS+2(R8)                 ; Initialize portion of RMS buffer
                         0421                                                  ;  descriptor for block I/O FTM
                         0421          $CONNECT RAB=R11                        ; Establish a record stream
             0A 50   E9  042A          BLBC    R0,10$                          ; Branch on failure
                         042D          $SETBIT #FAL$V_LAST_MSG,(R8)            ; Declare this last message to block
              072E   30  0431          BSBW    SEND_ACK                        ; Send Acknowledge message
              08AF   31  0434          BRW     EXIT_SUCCESS                    ; Exit state with success
              085D   31  0437   10$:   BRW     ERR_FILE_XFER                   ; Return error in Status message
                         043A
                         043A
                         043A          .SBTTL  FAL$CHECK_FTM
                         043A
                         043A   ;++
                         043A   ; This routine determines whether or not DAP file transfer mode has been
                         043A   ; specified. The file transfer mode flag stored in the FAL work area is used
                         043A   ; to form the state transition status code.
                         043A   ;--
                         043A
                         043A   FAL$CHECK_FTM::                                ; Entry point
  50   68   01   08  EF  043A          EXTZV   #FAL$V_FTM,#1,(R8),R0           ; Form yes/no reply
                     05  043F          RSB                                     ; Exit state with status code in R0
```

```
                        0440            .SBTTL  FAL$RETRV_RAM, FAL$RETRV_FTM
                        0440
                        0440    ;++
                        0440    ; These routines perform both DAP GET record and DAP READ block functions
                        0440    ; (also called the record/block retrieval functions in the DAP specification).
                        0440    ;
                        0440    ; This section includes two action routine entry points (which together
                        0440    ; transfer control to four action routines) and support routines used by the
                        0440    ; action routines.
                        0440    ;
                        0440    ; FAL$RETRV_RAM and FAL$RETRV_FTM are entry points referenced in the
                        0440    ; state transition table which dispatch to four related but distinct input
                        0440    ; action routines that perform the requested data retrieval operation. Each
                        0440    ; of these routines is specialized to use a combination of either record I/O
                        0440    ; or block I/O in either record access mode (RAM) or file transfer mode (FTM).
                        0440    ;--
                        0440
                        0440    FAL$RETRV_RAM::                                 ; Entry point
       03 68    09  E0  0440            BBS     #FAL$V_BLK_IO,(R8),10$          ; Branch if block I/O access
          0059      31  0444            BRW     FAL$GET_RAM                     ; This is a GET in record access mode
          0131      31  0447    10$:    BRW     FAL$READ_RAM                    ; This is a READ in record access mode
                        044A
                        044A    FAL$RETRV_FTM::                                 ; Entry point
       03 68    09  E0  044A            BBS     #FAL$V_BLK_IO,(R8),10$          ; Branch if block I/O access
          00CA      31  044E            BRW     FAL$GET_FTM                     ; This is a GET in file transfer mode
          0174      31  0451    10$:    BRW     FAL$READ_FTM                    ; This is a READ in file transfer mode
                        0454
                        0454    ;+
                        0454    ; Compute new cumulative CRC checksum value for the file based on the current
                        0454    ; record/block and the previous cumulative CRC value (if the CRC checking
                        0454    ; option has been requested by the partner process).
                        0454    ;
                        0454    ; Note that this routine is called only from the GET and READ code paths.
                        0454    ;
                        0454    ; On input <R4,R6> is descriptor of user record/block.
                        0454    ; On output R0-R3 is destroyed and DAP$L_CRC_RSLT is updated.
                        0454    ;-
                        0454
                        0454    COMPUTE_XMT_CRC:                                ; Entry point
                03  E1  0454            BBC     #DAP$V_RET_CRC,-                ; Branch if checksum option not
          0C 01F5 C8    0456                    FAL$B_ACCOPT(R8),10$            ;  requested by accessing node
          0000'CF   0B  045A            CRC     W^FAL$CRC_TABLE,-               ; Compute CRC value (destroying R0-R3)
             20 A9      045E                    DAP$L_CRC_RSLT(R9),-            ;  using result of previous CRC value
          66    54      0460                    R4,(R6)                         ;  calculation as initial CRC value
       20 A9    50  D0  0462            MOVL    R0,DAP$L_CRC_RSLT(R9)           ; Store resultant CRC value
                        0466
                        0466    ;
                        0466    ; Update statistics relative to the transmission of user data to the partner
                        0466    ; process.
                        0466    ;
                        0466
    57    00C0 C8   DE  0466    10$:    MOVAL   FAL$L_STB(R8),R7                ; Get address of Statistics Block
             1C A7  D6  046B            INCL    FAL$L_XMT_DAT(R7)               ; Increment XMT record/block count
       20 A7    54  C0  046E            ADDL2   R4,FAL$L_XMT_USR(R7)            ; Update XMT user data byte count
                    05  0472            RSB                                     ; Exit
```

```
                    0473  ;+
                    0473  ; Perform record terminator processing if the record format of the file is
                    0473  ; one of the three flavors of stream (i.e., stream, stream_LF, or stream_CR).
                    0473  ;
                    0473  ; On input R3 is address of last byte in record + 1.
                    0473  ; On output R0 is destroyed and R3 is updated if a terminator is appended.
                    0473  ;-
                    0473
                    0473  CHECK_STREAM:                                   ; Entry point
                    0473
                    0473          ASSUME  FAB$C_UDF EQ 0
                    0473          ASSUME  FAB$C_FIX EQ 1
                    0473          ASSUME  FAB$C_VAR EQ 2
                    0473          ASSUME  FAB$C_VFC EQ 3
                    0473          ASSUME  FAB$C_STM EQ 4
                    0473          ASSUME  FAB$C_STMLF EQ 5
                    0473          ASSUME  FAB$C_STMCR EQ 6
                    0473
                    0473          $CASEB  SELECTOR=FAB$B_RFM(R10)-  ;Dispatch on record format
                    0473                  BASE=#FAB$C_STM-          ;
                    0473                  DISPL=<-                  ; Record format:
                    0473                          10$-              ;  STM
                    0473                          40$-              ;  STMLF
                    0473                          50$-              ;  STMCR
                    0473                  >                         ;  UDF, FIX, VAR, and VFC
               05   047E          RSB                               ; Exit
                    047F
                    047F  ;
                    047F  ; Stream format--append CRLF if no valid terminator character found.
                    047F  ;
                    047F
            54 D5   047F  10$:    TSTL    R4                        ; Branch if null record
            0F 13   0481          BEQL    20$                       ;
     50  FF A3 9A   0483          MOVZBL  -1(R3),R0                 ; Get record terminator character
        1F  50 91   0487          CMPB    R0,#31                    ; Branch if out-of-range for a stream
            06 1A   048A          BGTRU   20$                       ;  terminator character
05 0000'CF  50 E0   048C          BBS     R0,W^FAL$STM_MASK,30$     ; Branch on valid stream terminator
   83  0A0D 8F B0   0492  20$:    MOVW    #CRLF,(R3)+               ; Add CRLF to terminate record
               05   0497  30$:    RSB                               ; Exit
                    0498
                    0498  ;
                    0498  ; Stream_LF format--append LF to record.
                    0498  ;
                    0498
        83  0A 90   0498  40$:    MOVB    #LF,(R3)+                 ; Add LF to terminate record
               05   049B          RSB                               ; Exit
                    049C
                    049C  ;
                    049C  ; Stream_CR format--append CR to record.
                    049C  ;
                    049C
        83  0D 90   049C  50$:    MOVB    #CR,(R3)+                 ; Add CR to terminate record
               05   049F          RSB                               ; Exit
```

```
                   04A0          .SBTTL  FAL$GET_RAM subsection
                   04A0
                   04A0  ;+
                   04A0  ; Perform the DAP GET record function in record access mode. This mode
                   04A0  ; requires that an explicit Status message be returned with each Data message.
                   04A0  ;-
                   04A0
                   04A0  FAL$GET_RAM::                                   ; GET specific code segment
        57  53  D0 04A0          MOVL    R3,R7                           ; Save address of start of buffer
                   04A3
                   04A3  ;
                   04A3  ; Build and send Data message to partner.
                   04A3  ; If VFC format, record header is prefixed to the record in one data field.
                   04A3  ;
                   04A3
     04 69  24  E0 04A3          BBS     #DAP$V_GEQ_V56,(R9),10$ ; Branch if partner uses DAP since V5.6
                   04A7          $SETBIT #FAL$V_LAST_MSG,(R8)    ; Declare this last message to block
        50  08  D0 04AB  10$:    MOVL    #DAP$K_DAT_MSG,R0       ; Get message type value
          FB4F' 30 04AE          BSBW    FAL$BUILD_HEAD          ; Construct message header
            83  94 04B1          CLRB    (R3)+                   ; Do not return record number
        56  53  D0 04B3          MOVL    R3,R6                   ; Save current address (i.e., start of
                   04B6                                          ;  DAP FILEDATA field in message)
     03  1F AA  91 04B6          CMPB    FAB$B_RFM(R10),#FAB$C_VFC
            0B  12 04BA          BNEQ    20$                     ; Branch if not VFC format
     2C AB  53  D0 04BC          MOVL    R3,RAB$L_RHB(R11)       ; Store address of record header buffer
     50  3F AA  9A 04C0          MOVZBL  FAB$B_FSZ(R10),R0       ; Get size of record header
        53  50  C0 04C4          ADDL2   R0,R3                   ; Skip over RHB storage area in DAP msg
     24 AB  53  D0 04C7  20$:    MOVL    R3,RAB$L_UBF(R11)       ; Store buffer address
  51    53  57  C3 04CB          SUBL3   R7,R3,R1                ; Compute # bytes preceding data record
     1A A8  51  A3 04CF          SUBW3   R1,FAL$W_DAPBUFSIZ(R8),-; Compute max # bytes that can be read
        20 AB      04D3                  RAB$W_USZ(R11)          ;  into rest of buffer
     04  1F AA  91 04D5          CMPB    FAB$B_RFM(R10),#FAB$C_STM
            04  1F 04D9          BLSSU   30$                     ; Branch if not STM, STMLF, or STMCR
     20 AB  02  A2 04DB          SUBW2   #2,RAB$W_USZ(R11)       ; Reserve space in message buffer for
                   04DF                                          ;  subsequent addition of terminator(s)
                   04DF  30$:    $GET    RAB=R11                 ; Get the record
         2D 50  E9 04E8          BLBC    R0,50$                  ; Branch on failure
     54  22 AB  3C 04EB          MOVZWL  RAB$W_RSZ(R11),R4       ; Get record size
        53  54  C0 04EF          ADDL2   R4,R3                   ; Update next byte pointer
          FF7E  30 04F2          BSBW    CHECK_STREAM            ; Process stream file terminator
  54    53  56  C3 04F5          SUBL3   R6,R3,R4                ; Compute size of FILEDATA field
          FB04' 30 04F9          BSBW    FAL$BUILD_TAIL          ; Finish building message
          FF55  30 04FC          BSBW    COMPUTE_XMT_CRC         ; Compute CRC value (destroying R0-R3)
                   04FF                                          ;  and update XMT data statistics
          FAFE' 30 04FF          BSBW    FAL$TRANSMIT            ; Send Data message
                   0502
                   0502  ;
                   0502  ; Send Status message to partner with RFA and BKT fields.
                   0502  ;
                   0502
     0F 69  24  E1 0502          BBC     #DAP$V_GEQ_V56,(R9),40$ ; Branch if partner uses DAP before V5.6
                   0506          $SETBIT #FAL$V_RET_RFA,(R8)     ; Return RFA of record
                   050A          $SETBIT #FAL$V_RET_RECN,(R8)    ; Return record number (meaningful for
                   050E                                          ;  REL file or fixed length SEQ file)
     50  08 AB  D0 050E          MOVL    RAB$L_STS(R11),R0       ; Restore success code
          07C4  31 0512          BRW     EXIT_STS_SUC            ; Convey success in Status message
          07CE  31 0515  40$:    BRW     EXIT_SUCCESS            ; Exit state with success
```

```
077C   31  0518  1108 50$:    BRW     ERR_FILE_XFER          ; Return error in Status message
```

```
                    051B                .SBTTL  FAL$GET_FTM subsection
                    051B
                    051B        ;+
                    051B        ; Perform the DAP GET record function in file transfer mode. This mode
                    051B        ; allows Data messages to be blocked together and pipelined on return.
                    051B        ;-
                    051B
                    051B        FAL$GET_FTM::                                   ; GET specific code segment
        57    53  D0 051B                MOVL    R3,R7                          ; Save address of start of buffer
                    051E
                    051E        ;
                    051E        ; Build and send Data message to partner.
                    051E        ; If VFC format, record header is prefixed to the record in one data field.
                    051E        ;
                    051E
        50    08  D0 051E                MOVL    #DAP$K_DAT_MSG,R0              ; Get message type value
           FADC'  30 0521                BSBW    FAL$BUILD_HEAD                 ; Construct message header
              83  94 0524                CLRB    (R3)+                          ; Do not return record number
        56    53  D0 0526                MOVL    R3,R6                          ; Save current address (i.e., start of
                    0529                                                        ;  DAP FILEDATA field in message)
     03   1F AA  91 0529                CMPB    FAB$B_RFM(R10),#FAB$C_VFC
              0B  12 052D                BNEQ    10$                            ; Branch if not VFC format
     2C AB    53  D0 052F                MOVL    R3,RAB$L_RHB(R11)              ; Store address of record header buffer
     50   3F AA  9A 0533                MOVZBL  FAB$B_FSZ(R10),R0              ; Get size of record header
        53    50  C0 0537                ADDL2   R0,R3                          ; Skip over RHB storage area in DAP msg
     24 AB    53  D0 053A        10$:    MOVL    R3,RAB$L_UBF(R11)              ; Store buffer address
  51    53    57  C3 053E                SUBL3   R7,R3,R1                       ; Compute # bytes preceding data record
     1A A8    51  A3 0542                SUBW3   R1,FAL$W_DAPBUFSIZ(R8),-       ; Compute max # bytes that can be read
        20 AB        0546                        RAB$W_USZ(R11)                 ;  into rest of buffer
     04   1F AA  91 0548                CMPB    FAB$B_RFM(R10),#FAB$C_STM
              04  1F 054C                BLSSU   20$                            ; Branch if not STM, STMLF, or STMCR
     20 AB    02  A2 054E                SUBW2   #2,RAB$W_USZ(R11)              ; Reserve space in message buffer for
                    0552                                                        ;  subsequent addition of terminator(s)
                    0552        20$:    $GET    RAB=R11                         ; Get the record
        1A    50  E9 055B                BLBC    R0,30$                         ; Branch on failure
     54   22 AB  3C 055E                MOVZWL  RAB$W_RSZ(R11),R4              ; Get record size
        53    54  C0 0562                ADDL2   R4,R3                          ; Update next byte pointer
            FF0B  30 0565                BSBW    CHECK_STREAM                   ; Process stream file terminator
  54    53    56  C3 0568                SUBL3   R6,R3,R4                       ; Compute size of FILEDATA field
           FA91'  30 056C                BSBW    FAL$BUILD_TAIL                 ; Finish building message
            FEE2  30 056F                BSBW    COMPUTE_XMT_CRC                ; Compute CRC value (destroying R0-R3)
                    0572                                                        ;  and update XMT data statistics
           FA8B'  30 0572                BSBW    FAL$TRANSMIT                   ; Send Data message
            076E  31 0575                BRW     EXIT_SUCCESS                   ; Exit state with success
            071C  31 0578        30$:    BRW     ERR_FILE_XFER                  ; Return error in Status message
```

```
                     057B  1156          .SBTTL  FAL$READ_RAM subsection
                     057B  1157
                     057B  1158 ;+
                     057B  1159 ; Perform the DAP READ block function in record access mode. This mode
                     057B  1160 ; requires that an explicit Status message be returned with each Data message.
                     057B  1161 ;-
                     057B  1162
                     057B  1163 FAL$READ_RAM::                                  ; READ specific code segment
                     057B  1164
                     057B  1165 ;
                     057B  1166 ; Build and send Data message to partner.
                     057B  1167 ;
                     057B  1168
04 69    24    E0    057B  1169          BBS     #DAP$V_GEQ_V56,(R9),10$ ; Branch if partner uses DAP since V5.6
                     057F  1170          $SETBIT #FAL$V_LAST_MSG,(R8)   ; Declare this last message to block
   50    08    D0    0583  1171 10$:     MOVL    #DAP$K_DAT_MSG,R0      ; Get message type value
      FA77'    30    0586  1172          BSBW    FAL$BUILD_HEAD         ; Construct message header
         83    94    0589  1173          CLRB    (R3)+                  ; Do not return virtual block number
   56    53    D0    058B  1174          MOVL    R3,R6                  ; Save current address (i.e., start of
                     058E  1175                                         ;  DAP FILEDATA field in message)
24 AB    53    D0    058E  1176          MOVL    R3,RAB$L_UBF(R11)      ; Store buffer address
   3C AA       B0    0592  1177          MOVW    FAB$W_BLS(R10),-       ; Request exactly one block
   20 AB             0595  1178                  RAB$W_USZ(R11)         ;
                     0597  1179          $READ   RAB=R11                ; Read the block
   22 50       E9    05A0  1180          BLBC    R0,30$                 ; Branch on failure
54 22 AB       3C    05A3  1181          MOVZWL  RAB$W_RSZ(R11),R4      ; Get number of bytes read
   53    54    C0    05A7  1182          ADDL2   R4,R3                  ; Update next byte pointer
      FA53'    30    05AA  1183          BSBW    FAL$BUILD_TAIL         ; Finish building message
      FEA4     30    05AD  1184          BSBW    COMPUTE_XMT_CRC        ; Compute CRC value (destroying R0-R3)
                     05B0  1185                                         ;  and update XMT data statistics
      FA4D'    30    05B0  1186          BSBW    FAL$TRANSMIT           ; Send Data message
                     05B3  1187
                     05B3  1188 ;
                     05B3  1189 ; Send Status message to partner with RFA field.
                     05B3  1190 ;
                     05B3  1191
0B 69    24    E1    05B3  1192          BBC     #DAP$V_GEQ_V56,(R9),20$ ; Branch if partner uses DAP before V5.6
                     05B7  1193          $SETBIT #FAL$V_RET_RFA,(R8)    ; Return RFA of record
50    08 AB    D0    05BB  1194          MOVL    RAB$L_STS(R11),R0      ; Restore success code
      0717     31    05BF  1195          BRW     EXIT_STS_SUC           ; Convey success in Status message
      0721     31    05C2  1196 20$:     BRW     EXIT_SUCCESS           ; Exit state with success
      06CF     31    05C5  1197 30$:     BRW     ERR_FILE_XFER          ; Return error in Status message
```

```
                           05C8          .SBTTL  FAL$READ_FTM subsection
                           05C8
                           05C8  ;+
                           05C8  ; Perform the DAP READ block function in file transfer mode. This mode
                           05C8  ; allows Data messages to be blocked together and pipelined on return.
                           05C8  ;-
                           05C8
                           05C8  FAL$READ_FTM::                                 ; READ specific code segment
       0200 8F    3C AA  B1  05C8          CMPW    FAB$W_BLS(R10),#512        ; Branch if we're using a nonstandard
                     04  12  05CE          BNEQ    10$                        ;  block size
            68       32  E1  05D0          BBC     #FAL$V_DIS_RBK,(R8),-      ; Are multi-block reads desired?
                     03      05D3                  READ_FTM_RBK               ;  Yes, take the normal path
                   0061  31  05D4  10$:    BRW     READ_FTM_NORBK             ; No, read one block at a time
                           05D7
                           05D7  ;+
                           05D7  ; This code path utilizes multi-block reads from disk where the buffer size
                           05D7  ; is determined by the value in FAL$B_RBK_CACHE. Note that use of this RMS
                           05D7  ; multi_block cache optimization for block I/O file transfer mode retrieval
                           05D7  ; necessitates an additional copy of the data in memory to be performed.
                           05D7  ;-
                           05D7
                           05D7  READ_FTM_RBK:                                  ; Enable buffering of RMS blocks
                  66 A8  B5  05D7          TSTW    FAL$Q_RMS+2(R8)            ; Branch if there is data in RMS
                     23  12  05DA          BNEQ    10$                        ;  buffer to process
                  64 A8  B0  05DC          MOVW    FAL$Q_RMS(R8),-            ; Store buffer size
                  20 AB      05DF                  RAB$W_USZ(R11)             ;
                  68 A8  D0  05E1          MOVL    FAL$Q_RMS+4(R8),-          ; Store buffer address
                  24 AB      05E4                  RAB$L_UBF(R11)             ;
                           05E6          $READ   RAB=R11                    ; Read specified number of blocks
                  43 50  E9  05EF          BLBC    R0,30$                     ; Branch on failure
                  38 AB  D4  05F2          CLRL    RAB$L_BKT(R11)             ; Zero VBN value to imply use of NBP
                           05F5                                             ;  in case Control message initiated
                           05F5                                             ;  file transfer with a non-zero VBN
                  22 AB  B0  05F5          MOVW    RAB$W_RSZ(R11),-           ; Save number of bytes actually read
                  66 A8      05F8                  FAL$Q_RMS+2(R8)            ;  into buffer
                  68 A8  D0  05FA          MOVL    FAL$Q_RMS+4(R8),-          ; Initialize next byte pointer to
                  6C A8      05FD                  FAL$L_RMS_PTR(R8)          ;  beginning of buffer
                           05FF
                           05FF  ;
                           05FF  ; Build and send Data message to partner.
                           05FF  ;
                           05FF
         54       3C AA  3C  05FF  10$:    MOVZWL  FAB$W_BLS(R10),R4          ; Get block size
         54       66 A8  B1  0603          CMPW    FAL$Q_RMS+2(R8),R4         ; Is there at least one full block left?
                     04  1E  0607          BGEQU   20$                        ; Branch if yes
         54       66 A8  3C  0609          MOVZWL  FAL$Q_RMS+2(R8),R4         ; Otherwise get partial block size
      66 A8          54  A2  060D  20$:    SUBW2   R4,FAL$Q_RMS+2(R8)         ; Update remaining byte count
            50       08  D0  0611          MOVL    #DAP$K_DAT_MSG,R0          ; Get message type value
                  F9E9'  30  0614          BSBW    FAL$BUILD_HEAD             ; Construct message header
                     83  94  0617          CLRB    (R3)+                      ; Do not return virtual block number
            56       53  D0  0619          MOVL    R3,R6                      ; Save current address (i.e., start of
                           061C                                             ;  DAP FILEDATA field in message)
   63     6C B8      54  28  061C          MOVC3   R4,@FAL$L_RMS_PTR(R8),(R3)
          6C A8      51  D0  0621          MOVL    R1,FAL$L_RMS_PTR(R8)       ; Copy data and update next byte pointer
       54    53      56  C3  0625          SUBL3   R6,R3,R4                   ; Compute size of FILEDATA field
                  F9D4'  30  0629          BSBW    FAL$BUILD_TAIL             ; Finish building message
                   FE25  30  062C          BSBW    COMPUTE_XMT_CRC            ; Compute CRC value (destroying R0-R3)
```

```
                062F                                          ;  and update XMT data statistics
   F9CE'   30   062F                  BSBW    FAL$TRANSMIT    ; Send Data message
    06B1   31   0632                  BRW     EXIT_SUCCESS    ; Exit state with success
    065F   31   0635      30$:        BRW     ERR_FILE_XFER   ; Return error in Status message
                0638
                0638      ;+
                0638      ; This code path does not use the RMS multi-block cache optimization for
                0638      ; block I/O file transfer mode retrieval.
                0638      ;
                0638      ; Note that this represents the FAL functionality for the VMS V3.0 release.
                0638      ;-
                0638
                0638      READ_FTM_NORBK:                     ; Disable buffering of RMS blocks
                0638
                0638      ;
                0638      ; Build and send Data message to partner.
                0638      ;
                0638
 50    08  D0   0638                  MOVL    #DAP$K_DAT_MSG,R0       ; Get message type value
     F9C2' 30   063B                  BSBW    FAL$BUILD_HEAD          ; Construct message header
       83  94   063E                  CLRB    (R3)+                   ; Do not return virtual block number
 56    53  D0   0640                  MOVL    R3,R6                   ; Save current address (i.e., start of
                0643                                                  ;  DAP FILEDATA field in message)
 24 AB 53  D0   0643                  MOVL    R3,RAB$L_UBF(R11)       ; Store buffer address
    3C AA  B0   0647                  MOVW    FAB$W_BLS(R10),-        ; Request exactly one block
    20 AB       064A                          RAB$W_USZ(R11)          ;
                064C                  $READ   RAB=R11                 ; Read the block
    16 50  E9   0655                  BLBC    R0,10$                  ; Branch on failure
    38 AB  D4   0658                  CLRL    RAB$L_BKT(R11)          ; Zero VBN value to imply use of NBP
                065B                                                  ;  in case Control message initiated
                065B                                                  ;  file transfer at a non-zero VBN
 54 22 AB  3C   065B                  MOVZWL  RAB$W_RSZ(R11),R4       ; Get number of bytes read
 53    54  C0   065F                  ADDL2   R4,R3                   ; Update next byte pointer
    F99B'  30   0662                  BSBW    FAL$BUILD_TAIL          ; Finish building message
     FDEC  30   0665                  BSBW    COMPUTE_XMT_CRC         ; Compute CRC value (destroying R0-R3)
                0668                                                  ;  and update XMT data statistics
    F995'  30   0668                  BSBW    FAL$TRANSMIT            ; Send Data message
     0678  31   066B                  BRW     EXIT_SUCCESS            ; Exit state with success
     0626  31   066E      10$:        BRW     ERR_FILE_XFER           ; Return error in Status message
```

```
                    0671          .SBTTL  FAL$STORE_RAM, FAL$STORE_FTM
                    0671
                    0671  ;++
                    0671  ; These routines perform both DAP PUT record and DAP WRITE block functions
                    0671  ; (also called the record/block storage functions in the DAP specification).
                    0671  ;
                    0671  ; This section includes two action routine entry points (which together
                    0671  ; transfer control to four action routines) and support routines used by the
                    0671  ; action routines.
                    0671  ;
                    0671  ; FAL$STORE_RAM and FAL$STORE_FTM are entry points referenced in the
                    0671  ; state transition table which dispatch to four related but distinct output
                    0671  ; action routines that perform the requested data storage operation. Each
                    0671  ; of these routines is specialized to use a combination of either record I/O
                    0671  ; or block I/O in either record access mode (RAM) or file transfer mode (FTM).
                    0671  ;--
                    0671
                    0671  FAL$STORE_RAM::                                  ; Entry point
   03 68    09  E0  0671          BBS     #FAL$V_BLK_IO,(R8),10$           ; Branch if block I/O access
        002C    31  0675          BRW     FAL$PUT_RAM                      ; This is a PUT in record access mode
        00B8    31  0678  10$:    BRW     FAL$WRITE_RAM                    ; This is a WRITE in record access mode
                    067B
                    067B  FAL$STORE_FTM::                                  ; Entry point
   03 68    09  E0  067B          BBS     #FAL$V_BLK_IO,(R8),10$           ; Branch if block I/O access
        007C    31  067F          BRW     FAL$PUT_FTM                      ; This is a PUT in file transfer mode
        00E3    31  0682  10$:    BRW     FAL$WRITE_FTM                    ; This is a WRITE in file transfer mode
                    0685
                    0685  ;+
                    0685  ; Compute new cumulative CRC checksum value for the file based on the current
                    0685  ; record/block and the previous cumulative CRC value (if the CRC checking
                    0685  ; option has been requested by the partner process).
                    0685  ;
                    0685  ; Note that this routine is called only from the PUT and WRITE code paths,
                    0685  ; and from the FAL$UPDATE routine.
                    0685  ;
                    0685  ; On input <R4,R5> is descriptor of user record/block.
                    0685  ; On output R0-R3 is destroyed and DAP$L_CRC_RSLT is updated.
                    0685  ;-
                    0685
                    0685  COMPUTE_RCV_CRC:                                 ; Entry point
          03    E1  0685          BBC     #DAP$V_RET_CRC,-                 ; Branch if checksum option not
  0C 01F5 C8        0687                  FAL$B_ACCOPT(R8),10$             ;  requested by accessing node
     0000'CF    0B  068B          CRC     W^FAL$CRC_TABLE,-                ; Compute CRC value (destroying R0-R3)
       20 A9        068F                  DAP$L_CRC_RSLT(R9),-             ;  using result of previous CRC value
     65    54       0691                  R4,(R5)                          ;  calculation as initial CRC value
  20 A9    50   D0  0693          MOVL    R0,DAP$L_CRC_RSLT(R9)            ; Store resultant CRC value
                    0697
                    0697  ;
                    0697  ; Update statistics relative to the reception of user data from the partner
                    0697  ; process.
                    0697  ;
                    0697
 57   00C0 C8   DE  0697  10$:    MOVAL   FAL$L_STB(R8),R7                 ; Get address of Statistics Block
        08 A7   D6  069C          INCL    FAL$L_RCV_DAT(R7)                ; Increment RCV record/block count
  0C A7    54   C0  069F          ADDL2   R4,FAL$L_RCV_USR(R7)             ; Update RCV user data byte count
            05      06A3          RSB                                      ; Exit
```

```
                      06A4          .SBTTL  FAL$PUT_RAM subsection
                      06A4
                      06A4  ;+
                      06A4  ; Perform the DAP PUT record function in record access mode. This mode
                      06A4  ; requires that an explicit Status message be returned.
                      06A4  ;-
                      06A4
                      06A4  FAL$PUT_RAM::                                 ; PUT specific code segment
                      06A4
                      06A4  ;+
                      06A4  ; Process the DAP record number field of the Data message.
                      06A4  ; For random access by key value for a relative (or fixed length sequential)
                      06A4  ; file it contains the relative record number. Otherwise, it is not used.
                      06A4  ;-
                      06A4
   12 69    00   E1   06A4          BBC     #DAP$V_X_RECNUM,(R9),10$; Branch if DAP record number field
                      06A8                                           ;  was null (as opposed to specified
                      06A8                                           ;  with a zero value)
01    1E AB    91     06A8          CMPB    RAB$B_RAC(R11),#RAB$C_KEY;Branch if not random access
         0C    12     06AC          BNEQ    10$                      ;  by key value
20    1D AA    91     06AE          CMPB    FAB$B_ORG(R10),#FAB$C_IDX;Branch if IDX organization
         06    13     06B2          BEQL    10$                      ; Fall thru if SEQ or REL organization
      40 A9    D0     06B4          MOVL    DAP$L_RECNUM1(R9),-      ; Update relative record number
    01FC C8           06B7                  FAL$L_NUMBER(R8)         ;  for next PUT operation
                      06BA
                      06BA  ;+
                      06BA  ; Process the DAP FILEDATA field.
                      06BA  ; If VFC format, it contains the record header prefixed to the record.
                      06BA  ;-
                      06BA
54    44 A9    7D     06BA  10$:    MOVQ    DAP$Q_FILEDATA(R9),R4    ; Store record descriptor in <R4,R5>
       FFC4    30     06BE          BSBW    COMPUTE_RCV_CRC          ; Compute CRC value (destroying R0-R3)
                      06C1                                           ;  and update RCV data statistics
03    1F AA    91     06C1          CMPB    FAB$B_RFM(R10),#FAB$C_VFC
         0E    12     06C5          BNEQ    20$                      ; Branch if not VFC format
2C AB    55    D0     06C7          MOVL    R5,RAB$L_RHB(R11)        ; Store address of record header buffer
52    3F AA    9A     06CB          MOVZBL  FAB$B_FSZ(R10),R2        ; Get size of record header
   54    52    C2     06CF          SUBL2   R2,R4                    ; Compute size of record
   55    52    C0     06D2          ADDL2   R2,R5                    ; Compute address of record
22 AB    54    B0     06D5  20$:    MOVW    R4,RAB$W_RSZ(R11)        ; Store record size
28 AB    55    D0     06D9          MOVL    R5,RAB$L_RBF(R11)        ; Store record address
                      06DD          $PUT    RAB=R11                  ; Put the record
      12 50    E9     06E6          BLBC    R0,40$                   ; Branch on failure
                      06E9
                      06E9  ;
                      06E9  ; Send Status message to partner with RFA and BKT fields.
                      06E9  ;
                      06E9
0B 69    24    E1     06E9          BBC     #DAP$V_GEQ_V56,(R9),30$ ; Branch if partner uses DAP before V5.6
                      06ED          $SETBIT #FAL$V_RET_RFA,(R8)      ; Return RFA of record
                      06F1          $SETBIT #FAL$V_RET_RECN,(R8)     ; Return record number (meaningful for
                      06F5                                           ;  REL file or fixed length SEQ file)
       05E1    31     06F5          BRW     EXIT_STS_SUC             ; Convey success in Status message
       05EB    31     06F8  30$:    BRW     EXIT_SUCCESS             ; Exit state with success
       0599    31     06FB  40$:    BRW     ERR_FILE_XFER            ; Return error in Status message
```

```
                        06FE          .SBTTL  FAL$PUT_FTM subsection
                        06FE
                        06FE    ;+
                        06FE    ; Perform the DAP PUT record function in file transfer mode. This mode
                        06FE    ; allows Data messages received to be blocked together and pipelined.
                        06FE    ;-
                        06FE
                        06FE    FAL$PUT_FTM::                                 ; PUT specific code segment
                        06FE
                        06FE    ;+
                        06FE    ; Process the DAP FILEDATA field.
                        06FE    ; If VFC format, it contains the record header prefixed to the record.
                        06FE    ;-
                        06FE
54    44 A9     7D      06FE            MOVQ    DAP$Q_FILEDATA(R9),R4   ; Store record descriptor in <R4,R5>
      FF80      30      0702            BSBW    COMPUTE_RCV_CRC         ; Compute CRC value (destroying R0-R3)
                        0705                                            ;  and update RCV data statistics
03    1F AA     91      0705            CMPB    FAB$B_RFM(R10),#FAB$C_VFC
         0E     12      0709            BNEQ    10$                     ; Branch if not VFC format
2C AB    55     D0      070B            MOVL    R5,RAB$L_RHB(R11)       ; Store address of record header buffer
52    3F AA     9A      070F            MOVZBL  FAB$B_FSZ(R10),R2       ; Get size of record header
   54    52     C2      0713            SUBL2   R2,R4                   ; Compute size of record
   55    52     C0      0716            ADDL2   R2,R5                   ; Compute address of record
22 AB    54     B0      0719    10$:    MOVW    R4,RAB$W_RSZ(R11)       ; Store record size
28 AB    55     D0      071D            MOVL    R5,RAB$L_RBF(R11)       ; Store record address
                        0721            $PUT    RAB=R11                 ; Put the record
      03 50     E9      072A            BLBC    R0,20$                  ; Branch on failure
       05B6     31      072D            BRW     EXIT_SUCCESS            ; Exit state with success
       0564     31      0730    20$:    BRW     ERR_FILE_XFER           ; Return error in Status message
```

```
                           0733            .SBTTL  FAL$WRITE_RAM subsection
                           0733
                           0733  ;+
                           0733  ; Perform the DAP WRITE block function in record access mode. This mode
                           0733  ; requires that an explicit Status message be returned.
                           0733  ;-
                           0733
                           0733  FAL$WRITE_RAM::                                  ; WRITE specific code segment
                           0733
                           0733  ;+
                           0733  ; Process the DAP record number field of the Data message.
                           0733  ; For random block I/O access, if the RECNUM field of the Data message is
                           0733  ; explicitly specified, it supersedes the KEY field of the Control message.
                           0733  ;-
                           0733
05 69   00    E1           0733            BBC     #DAP$V_X_RECNUM,(R9),10$; Branch if DAP record number field
                           0737                                             ; was null (as opposed to specified
                           0737                                             ; with a zero value)
     40 A9    D0           0737            MOVL    DAP$L_RECNUM1(R9),-      ; Store virtual block number in BKT
     38 AB                 073A                    RAB$L_BKT(R11)           ;   field of RAB
                           073C
                           073C  ;+
                           073C  ; Process the DAP FILEDATA field.
                           073C  ;-
                           073C
54    44 A9   7D           073C  10$:      MOVQ    DAP$Q_FILEDATA(R9),R4    ; Store block descriptor in <R4,R5>
       FF42   30           0740            BSBW    COMPUTE_RCV_CRC          ; Compute CRC value (destroying R0-R3)
                           0743                                             ;  and update RCV data statistics
22 AB   54    B0           0743            MOVW    R4,RAB$W_RSZ(R11)        ; Store block size
28 AB   55    D0           0747            MOVL    R5,RAB$L_RBF(R11)        ; Store block address
                           074B            $WRITE  RAB=R11                  ; Write the block
     0E 50    E9           0754            BLBC    R0,30$                   ; Branch on failure
                           0757
                           0757  ;
                           0757  ; Send Status message to partner with RFA field.
                           0757  ;
                           0757
07 69   24    E1           0757            BBC     #DAP$V_GEQ_V56,(R9),20$ ; Branch if partner uses DAP before V5.6
                           075B            $SETBIT #FAL$V_RET_RFA,(R8)     ; Return RFA of record
       0577   31           075F            BRW     EXIT_STS_SUC             ; Convey success in Status message
       0581   31           0762  20$:      BRW     EXIT_SUCCESS             ; Exit state with success
       052F   31           0765  30$:      BRW     ERR_FILE_XFER            ; Return error in Status message
```

```
                        0768        .SBTTL  FAL$WRITE_FTM subsection
                        0768
                        0768  ;+
                        0768  ; Perform the DAP WRITE block function in file transfer mode. This mode
                        0768  ; allows Data messages received to be blocked together and pipelined.
                        0768  ;-
                        0768
                        0768  FAL$WRITE_FTM::                         ; WRITE specific code segment
                        0768
                        0768  ;+
                        0768  ; Process the DAP FILEDATA field.
                        0768  ;-
                        0768
      54    44 A9   7D  0768          MOVQ    DAP$Q_FILEDATA(R9),R4  ; Store block descriptor in <R4,R5>
              FF16  30  076C          BSBW    COMPUTE_RCV_CRC        ; Compute CRC value (destroying R0-R3)
                        076F                                         ;  and update RCV data statistics
 0200 8F    3C AA   B1  076F          CMPW    FAB$W_BLS(R10),#512    ; Branch if we're using a nonstandard
                04  12  0775          BNEQ    10$                    ;  block size
        68      32  E1  0777          BBC     #FAL$V_DIS_RBK,(R8),-  ; Are multi-block writes desired?
                03      077A                  WRITE_FTM_RBK          ;  Yes, take the normal path
              0064  31  077B  10$:    BRW     WRITE_FTM_NORBK        ;  No, write one block at a time
                        077E
                        077E  ;+
                        077E  ; This code path utilizes multi-block writes to disk where the buffer size
                        077E  ; is determined by the value in FAL$B_RBK_CACHE. Note that use of this RMS
                        077E  ; multi_block cache optimization for block I/O file transfer mode storage
                        077E  ; necessitates an additional copy of the data in memory to be performed.
                        077E  ;-
                        077E
                        077E  WRITE_FTM_RBK:                         ; Enable buffering of RMS blocks
                00  E1  077E          BBC     #RAB$V_ASY,-           ; Branch if no asynchronous write
          1A 04 AB      0780                  RAB$L_ROP(R11),20$     ;  operation requires a wait call
             08 AB  B5  0783          TSTW    RAB$L_STS(R11)         ; Increment wait counter if write
                04  12  0786          BNEQ    10$                    ;  operation is still pending (but
           0000'CF  D6  0788          INCL    W^FAL$GL_WRITWAIT      ;  issue wait request regardless)
                        078C  10$:    $WAIT   RAB=R11                ; Wait for asynchronous write to finish
                        0795          $CLRBIT #RAB$V_ASY,RAB$L_ROP(R11);Clear asynchronous operation flag
             42 50  E9  079A          BLBC    R0,40$                 ; Branch on write failure
      66 A8     54  A0  079D  20$:    ADDW2   R4,FAL$Q_RMS+2(R8)     ; Update byte count
 6C B8    65    54  28  07A1          MOVC3   R4,(R5),@FAL$L_RMS_PTR(R8)
      6C A8     53  D0  07A6          MOVL    R3,FAL$L_RMS_PTR(R8)   ; Copy data and update next byte pointer
         66 A8      A3  07AA          SUBW3   FAL$Q_RMS+2(R8),-      ; Compute number of unused bytes in RMS
      50 64 A8          07AD                  FAL$Q_RMS(R8),R0       ;  buffer
      3C AA     50  B1  07B0          CMPW    R0,FAB$W_BLS(R10)      ; Branch if there is at least one full
                26  1E  07B4          BGEQU   30$                    ;  block left in RMS buffer
             66 A8  B0  07B6          MOVW    FAL$Q_RMS+2(R8),-      ; Store buffer size
             22 AB      07B9                  RAB$W_RSZ(R11)         ;
             68 A8  D0  07BB          MOVL    FAL$Q_RMS+4(R8),-      ; Store buffer address
             28 AB      07BE                  RAB$L_RBF(R11)         ;
                        07C0          $SETBIT #RAB$V_ASY,RAB$L_ROP(R11);Set asynchronous operation flag
                        07C5          $WRITE  RAB=R11                ; Write specified number of blocks
             0E 50  E9  07CE          BLBC    R0,40$                 ; Branch on failure
             38 AB  D4  07D1          CLRL    RAB$L_BKT(R11)         ; Zero VBN value to imply use of NBP
                        07D4                                         ;  in case Control message initiated
                        07D4                                         ;  file transfer with a non-zero VBN
             66 A8  B4  07D4          CLRW    FAL$Q_RMS+2(R8)        ; Initialize byte counter
             68 A8  D0  07D7          MOVL    FAL$Q_RMS+4(R8),-      ; Initialize next byte pointer
```

```
  6C AB         07DA                        FAL$L_RMS_PTR(R8)       ;
   0507   31    07DC  30$:          BRW     EXIT_SUCCESS            ; Exit state with success
   04B5   31    07DF  40$:          BRW     ERR_FILE_XFER           ; Return error in Status message
                07E2
                07E2  ;+
                07E2  ; This code path does not use the RMS multi-block cache optimization for
                07E2  ; block I/O file transfer mode storage.
                07E2  ;
                07E2  ; Note that this represents the FAL functionality for the VMS V3.0 release.
                07E2  ;-
                07E2
                07E2  WRITE_FTM_NORBK:                              ; Disable buffering of RMS blocks
22 AB    54   B0    07E2            MOVW    R4,RAB$W_RSZ(R11)       ; Store block size
28 AB    55   D0    07E6            MOVL    R5,RAB$L_RBF(R11)       ; Store block address
                07EA                $WRITE  RAB=R11                 ; Write the block
  06 50       E9    07F3            BLBC    R0,10$                  ; Branch on failure
  38 AB       D4    07F6            CLRL    RAB$L_BKT(R11)          ; Zero VBN value to imply use of NBP
                07F9                                                ;  in case Control message initiated
                07F9                                                ;  file transfer at a non-zero VBN
   04EA   31    07F9                BRW     EXIT_SUCCESS            ; Exit state with success
   0498   31    07FC  10$:          BRW     ERR_FILE_XFER           ; Return error in Status message
```

```
                      07FF           .SBTTL  FAL$STORE_END
                      07FF
                      07FF   ;++
                      07FF   ; This routine completes any block I/O activity in progress in order to
                      07FF   ; terminate file transfer mode.
                      07FF   ;--
                      07FF
                      07FF   FAL$STORE_END::                          ; Entry point
37 68      09    E1   07FF           BBC     #FAL$V_BLK_IO,(R8),20$   ; Branch if record mode access
33 68      32    E0   0803           BBS     #FAL$V_DIS_RBK,(R8),20$  ; Branch if RMS buffering is disabled
      66 A8      B5   0807           TSTW    FAL$Q_RMS+2(R8)          ; Branch if there is data in RMS
         18      12   080A           BNEQ    10$                      ;  buffer to write to disk
                      080C
                      080C   ;+
                      080C   ; Wait for previous write request to complete.
                      080C   ;-
                      080C
           00    E1   080C           BBC     #RAB$V_ASY,-             ; Branch if no asynchronous write
   29 04 AB           080E                   RAB$L_ROP(R11),20$       ;  operation requires a wait call
                      0811           $WAIT   RAB=R11                  ; Wait for asynchronous write to finish
                      081A           $CLRBIT #RAB$V_ASY,RAB$L_ROP(R11);Clear asynchronous operation flag
      1B 50      E9   081F           BLBC    R0,30$                   ; Branch on write failure
         16      11   0822           BRB     20$                      ; Branch on success
                      0824
                      0824   ;+
                      0824   ; Write out blocks already moved to the RMS buffer.
                      0824   ;-
                      0824
      66 A8      B0   0824   10$:    MOVW    FAL$Q_RMS+2(R8),-        ; Store buffer size
      22 AB           0827                   RAB$W_RSZ(R11)           ;
      68 A8      D0   0829           MOVL    FAL$Q_RMS+4(R8),-        ; Store buffer address
      28 AB           082C                   RAB$L_RBF(R11)           ;
                      082E           $WRITE  RAB=R11                  ; Write specified number of blocks
      03 50      E9   0837           BLBC    R0,30$                   ; Branch on failure
       04A9      31   083A   20$:    BRW     EXIT_SUCCESS             ; Exit state with success
       0457      31   083D   30$:    BRW     ERR_FILE_XFER            ; Return error in Status message
```

```
                        0840           .SBTTL  FAL$BIT_BUCKET
                        0840
                        0840   ;++
                        0840   ; This routine throws away any data that may be stored in the RMS buffer
                        0840   ; to support the skip record/block error recovery option for file transfer
                        0840   ; storage mode. (Note that the data has already been CRC checked when the
                        0840   ; Data message was processed, and the data buffered in memory).
                        0840   ;--
                        0840
                        0840   FAL$BIT_BUCKET::                                ; Entry point
             66 A8  B4  0840           CLRW    FAL$Q_RMS+2(R8)         ; Zero size in RMS buffer descriptor
              04A0  31  0843           BRW     EXIT_SUCCESS            ; Exit state with success
                        0846
                        0846
                        0846           .SBTTL  FAL$DISCARD_DAT
                        0846
                        0846   ;++
                        0846   ; This routine computes the CRC checksum of the record/block just received
                        0846   ; and then throws away the data. It is called during file transfer storage
                        0846   ; mode error recovery where data in the pipe is discarded until (but CRC
                        0846   ; checked) until a synchronization point is reached (typically an Access
                        0846   ; Complete message is received).
                        0846   ;--
                        0846
                        0846   FAL$DISCARD_DAT::                               ; Entry point
          54 44 A9  7D  0846           MOVQ    DAP$Q_FILEDATA(R9),R4   ; Store block descriptor in <R4,R5>
              FE38  30  084A           BSBW    COMPUTE_RCV_CRC         ; Compute CRC value (destroying R0-R3)
                        084D                                           ;  and update RCV data statistics
              0496  31  084D           BRW     EXIT_SUCCESS            ; Exit state with success
```

```
                     0850          .SBTTL  FAL$UPDATE
                     0850
                     0850    ;++
                     0850    ; This routine performs the DAP UPDATE record function.
                     0850    ;--
                     0850
                     0850    FAL$UPDATE::                              ; Entry point
                     0850
                     0850    ;+
                     0850    ; Process the DAP FILEDATA field.
                     0850    ; If VFC format, it contains the record header prefixed to the record.
                     0850    ;-
                     0850
54    44 A9    7D    0850            MOVQ    DAP$Q_FILEDATA(R9),R4     ; Store record descriptor in <R4,R5>
         FE2E  30    0854            BSBW    COMPUTE_RCV_CRC           ; Compute CRC value (destroying R0-R3)
                     0857                                              ;  and update RCV data statistics
03    1F AA    91    0857            CMPB    FAB$B_RFM(R10),#FAB$C_VFC
         0E    12    085B            BNEQ    10$                       ; Branch if not VFC format
2C AB    55    D0    085D            MOVL    R5,RAB$L_RHB(R11)         ; Store address of record header buffer
52    3F AA    9A    0861            MOVZBL  FAB$B_FSZ(R10),R2         ; Get size of record header
   54    52    C2    0865            SUBL2   R2,R4                     ; Compute size of record
   55    52    C0    0868            ADDL2   R2,R5                     ; Compute address of record
22 AB    54    B0    086B    10$:    MOVW    R4,RAB$W_RSZ(R11)         ; Store record size
28 AB    55    D0    086F            MOVL    R5,RAB$L_RBF(R11)         ; Store record address
                     0873            $UPDATE RAB=R11                   ; Update the record
      0E 50    E9    087C            BLBC    R0,30$                    ; Branch on failure
                     087F
                     087F    ;
                     087F    ; Send Status message to partner with RFA field.
                     087F    ;
                     087F
07 69    24    E1    087F            BBC     #DAP$V_GEQ_V56,(R9),20$   ; Branch if partner uses DAP before V5.6
                     0883            $SETBIT #FAL$V_RET_RFA,(R8)       ; Return RFA of record
       044F    31    0887            BRW     EXIT_STS_SUC              ; Convey success in Status message
       0459    31    088A    20$:    BRW     EXIT_SUCCESS              ; Exit state with success
       0407    31    088D    30$:    BRW     ERR_FILE_XFER             ; Return error in Status message
```

```
                    0890          .SBTTL  FAL$DELETE
                    0890
                    0890  ;++
                    0890  ; This routine performs the DAP DELETE record function.
                    0890  ;--
                    0890
                    0890  FAL$DELETE::                                    ; Entry point
                    0890          $DELETE RAB=R11                         ; Delete the record
     0A 50    E9    0899          BLBC    R0,20$                          ; Branch on failure
                    089C
                    089C  ;
                    089C  ; Send Status message to partner.
                    089C  ;
                    089C
  03 69   24  E1    089C          BBC     #DAP$V_GEQ_V56,(R9),10$ ; Branch if partner uses DAP before V5.6
     0436     31    08A0          BRW     EXIT_STS_SUC                    ; Convey success in Status message
     0440     31    08A3  10$:    BRW     EXIT_SUCCESS                    ; Exit state with success
     03EE     31    08A6  20$:    BRW     ERR_FILE_XFER                   ; Return error in Status message
```

```
                  08A9              .SBTTL  FAL$FIND
                  08A9
                  08A9  ;++
                  08A9  ; This routine performs the DAP FIND record function.
                  08A9  ;--
                  08A9
                  08A9  FAL$FIND::                                    ; Entry point
                  08A9              $FIND   RAB=R11                   ; Find the record
   1[illegible] 50   E9  08B2              BLBC    R0,20$                    ; Branch on failure
                  08B5
                  08B5  ;
                  08B5  ; Send Status message to partner with RFA and BKT fields.
                  08B5  ;
                  08B5
  0B 69   24   E1  08B5              BBC     #DAP$V_GEQ_V56,(R9),10$   ; Branch if partner uses DAP before V5.6
                  08B9              $SETBIT #FAL$V_RET_RFA,(R8)       ; Return RFA of record
                  08BD              $SETBIT #FAL$V_RET_RECN,(R8)      ; Return record number (meaningful for
                  08C1                                                ;  REL file or fixed length SEQ file)
     0415   31   08C1              BRW     EXIT_STS_SUC              ; Convey success in Status message
     041F   31   08C4  10$:        BRW     EXIT_SUCCESS              ; Exit state with success
     03CD   31   08C7  20$:        BRW     ERR_FILE_XFER             ; Return error in Status message
```

```
                    08CA          .SBTTL  FAL$DISPLAY
                    08CA
                    08CA  ;++
                    08CA  ; This routine performs the DAP DISPLAY file attributes function.
                    08CA  ;
                    08CA  ; Note: RMS returns only XAB information on $DISPLAY. The DAP specification,
                    08CA  ;       however, allows partner to request that the DAP Attributes and Name
                    08CA  ;       messages be returned in addition to Extended Attributes messages.
                    08CA  ;       If Attributes or Name messages are requested, they will be built from
                    08CA  ;       information in the FAB and NAM blocks, respectively, that were updated
                    08CA  ;       at open/create time.
                    08CA  ;--
                    08CA
                    08CA  FAL$DISPLAY::                                       ; Entry point
     24 AA    D4    08CA          CLRL    FAB$L_XAB(R10)                      ; Remove any XABs from chain
     24 AA    DE    08CD          MOVAL   FAB$L_XAB(R10),-                    ;
     7C A8          08D0                  FAL$L_CHAIN_NXT(R8)                 ; Save address of next chain pointer
      016B    30    08D2          BSBW    CHAIN_RQST_XABS                     ; Add required XABs to XAB chain
                    08D5          $DISPLAY FAB=R10                            ; Obtain attributes of the file
     13 50    E9    08DE          BLBC    R0,10$                              ; Branch on failure
      0193    30    08E1          BSBW    CHAIN_RQST_PRT2                     ; Fill in alterate ALLXABs and KEYXABs
                    08E4                                                      ;  as directed by NOA and NOK values
     0D 50    E9    08E4          BLBC    R0,10$                              ; Branch on failure
                    08E7
                    08E7  ;
                    08E7  ; Return (main) Attributes, Extended Attributes, and (resultant) Name messages
                    08E7  ; to partner as directed by request mask, terminated by an Acknowledge message.
                    08E7  ;
                    08E7
      0299    30    08E7          BSBW    SEND_OPTNL_MSGS                     ; Build and send optional DAP messages
                    08EA          $SETBIT #FAL$V_LAST_MSG,(R8)                ; Declare this last message to block
      0271    30    08EE          BSBW    SEND_ACK                            ; Send Acknowledge message
      03F2    31    08F1          BRW     EXIT_SUCCESS                        ; Exit state with success
      03A0    31    08F4  10$:    BRW     ERR_FILE_XFER                       ; Return error in Status message
```

```
                  08F7          .SBTTL  FAL$EXTEND
                  08F7
                  08F7     ;++
                  08F7     ; This routine performs the DAP EXTEND file allocation function.
                  08F7     ;--
                  08F7
                  08F7     FAL$EXTEND::                                  ; Entry point
   24 AA     D4   08F7            CLRL    FAB$L_XAB(R10)                 ; Remove any XABs from chain
   24 AA     DE   08FA            MOVAL   FAB$L_XAB(R10),-               ;
   7C A8          08FD                    FAL$L_CHAIN_NXT(R8)            ; Save address of next chain pointer
    01D1     30   08FF            BSBW    CHAIN_RECV_XABS                ; Add received XABs to XAB chain
                  0902            $EXTEND FAB=R10                        ; Extend the file
   0D 50     E9   090B            BLBC    R0,10$                         ; Branch on failure
                  090E
                  090E     ;
                  090E     ; Return Allocation message(s) terminated by an Acknowledge message.
                  090E     ;
                  090E
    0272     30   090E            BSBW    SEND_OPTNL_MSGS                ; Build and send optional DAP messages
                  0911            $SETBIT #FAL$V_LAST_MSG,(R8)           ; Declare this last message to block
    024A     30   0915            BSBW    SEND_ACK                       ; Send Acknowledge message
    03CB     31   0918            BRW     EXIT_SUCCESS                   ; Exit state with success
    0379     31   091B     10$:   BRW     ERR_FILE_XFER                  ; Return error in Status message
```

```
              091E          .SBTTL  FAL$REWIND
              091E          .SBTTL  FAL$TRUNCATE, FAL$FLUSH
              091E          .SBTTL  FAL$FREE, FAL$RELEASE
              091E
              091E  ;++
              091E  ; The following routines have separate entry points but share common code ...
              091E  ;
              091E  ; This routine performs the DAP REWIND file function.
              091E  ;--
              091E
              091E  FAL$REWIND::                                    ; Entry point
              091E          $REWIND RAB=R11                         ; Rewind the record stream
     43   11  0927          BRB     EXIT_STS_ALWAYS                 ; Join common code
              0929
              0929  ;++
              0929  ; This routine performs the DAP TRUNCATE file function.
              0929  ;--
              0929
              0929  FAL$TRUNCATE::                                  ; Entry point
              0929          $TRUNCATE RAB=R11                       ; Truncate the file
     38   11  0932          BRB     EXIT_STS_ALWAYS                 ; Join common code
              0934
              0934  ;++
              0934  ; This routine performs the DAP FLUSH file function.
              0934  ;--
              0934
              0934  FAL$FLUSH::                                     ; Entry point
              0934          $FLUSH RAB=R11                          ; Flush I/O buffers and write modified
              093D                                                  ;  file attributes
     2D   11  093D          BRB     EXIT_STS_ALWAYS                 ; Join common code
              093F
              093F  ;++
              093F  ; This routine performs the DAP FREE locked record function.
              093F  ;--
              093F
              093F  FAL$FREE::                                      ; Entry point
              093F          $FREE RAB=R11                           ; Free all locked records
     22   11  0948          BRB     EXIT_STS_ALWAYS                 ; Join common code
              094A
              094A  ;++
              094A  ; This routine performs the DAP RELEASE all locked records function.
              094A  ;--
              094A
              094A  FAL$RELEASE::                                   ; Entry point
              094A          $RELEASE RAB=R11                        ; Release specified locked record
     17   11  0953          BRB     EXIT_STS_ALWAYS                 ; Join common code
```

```
                0955          .SBTTL  FAL$SPACE_BW, FAL$SPACE_FW
                0955
                0955  ;++
                0955  ; These routines perform the DAP SPACE functions, space-forward and
                0955  ; space-backward.
                0955  ;
                0955  ; Note: They return the actual # blocks spaced as an unsigned number obtained
                0955  ;       from STV which may be non-zero for RMS$_NORMAL, RMS$_BOF, and RMS$_EOF
                0955  ;       completion codes.
                0955  ;--
                0955
                0955  FAL$SPACE_BW::                                  ; Entry point for space backward
   38 AB   CE   0955          MNEGL   RAB$L_BKT(R11),-                ; DAP sends an absolute value,
   38 AB        0958                  RAB$L_BKT(R11)                  ;  so negate it
                095A  FAL$SPACE_FW::                                  ; Entry point for space forward
                095A          $SPACE  RAB=R11                         ; Skip or backspace specified # blocks
                0963          $SETBIT #FAL$V_RET_RECN,(R8)            ; Always return actual # blocks spaced
                0967                                                  ;  as an unsigned number in RECNUM2
   0C AB   D0   0967          MOVL    RAB$L_STV(R11),-                ; Copy STV value to BKT field so that
   38 AB        096A                  RAB$L_BKT(R11)                  ;  STV value will be returned in DAP
                096C                                                  ;  RECNUM2 field per DAP spec
                096C
                096C  ;
                096C  ; Common exit path for $REWIND, $TRUNCATE, $FLUSH, $FREE, $RELEASE, and $SPACE.
                096C  ;
                096C
                096C  EXIT_STS_ALWAYS:                                ; Check status of RMS function call
   03 50   E9   096C          BLBC    R0,10$                          ; Branch on failure
                096F
                096F  ;
                096F  ; Send Status message to partner.
                096F  ;
                096F
    0367   31   096F          BRW     EXIT_STS_SUC                    ; Convey success in Status message
    0322   31   0972  10$:    BRW     ERR_FILE_XFER                   ; Return error in Status message
```

```
                    0975                 .SBTTL  FAL$DISCONNECT
                    0975
                    0975      ;++
                    0975      ; This routine performs the DAP DISCONNECT record stream function
                    0975      ; (also called the end-of-stream function in the DAP specification).
                    0975      ;--
                    0975
                    0975      FAL$DISCONNECT::                                      ; Entry point
                    0975                $DISCONNECT RAB=R11                         ; Destroy the record stream
      0A 50    E9   097E                BLBC    R0,20$                              ; Branch on failure
   03 69  21   E1   0981                BBC     #DAP$V_GEQ_V42,(R9),10$             ; Branch if partner uses DAP before V4.2
                    0985                                                            ;  (namely, DAP V4.1)
       01E7    30   0985                BSBW    SEND_CMP                            ; Send Access Complete message
       035B    31   0988      10$:      BRW     EXIT_SUCCESS                        ; Exit state with success
       0312    31   098B      20$:      BRW     ERR_FILE_CLOS                       ; Return error in Status message
```

```
                        098E             .SBTTL  FAL$RESET, FAL$CHANGE
                        098E
                        098E     ;++
                        098E     ; The following routines have separate entry points but share common code ...
                        098E     ;
                        098E     ; This routine performs the DAP RESET file function. RESET is similar to CLOSE
                        098E     ; except that the file is erased if it was created by this access operation.
                        098E     ;
                        098E     ; Note: Before DAP V6.0 this function was called PURGE.
                        098E     ;--
                        098E
                        098E     FAL$RESET::                                     ; Entry point
       01F6 C8    91    098E             CMPB    FAL$B_ACCFUNC(R8),-             ; Erase file if this was a
            02          0992                     #DAP$K_CREATE                   ;  file creation
            07    13    0993             BEQL    10$                             ;
       01F6 C8    91    0995             CMPB    FAL$B_ACCFUNC(R8),-             ;  or a
            07          0999                     #DAP$K_SUBMIT                   ;  file submission
            34    12    099A             BNEQ    CLOSE_COMMON                    ;  operation
                        099C     10$:    $SETBIT #FAB$V_DLT,FAB$L_FOP(R10);Set delete-on-close bit
            2D    11    09A1             BRB     CLOSE_COMMON                    ; Join common close code
                        09A3
                        09A3     ;++
                        09A3     ; This routine performs the DAP CHANGE file attributes on close function.
                        09A3     ;--
                        09A3
                        09A3     FAL$CHANGE::                                    ; Entry point
                        09A3
                        09A3     ;
                        09A3     ; Protection and Revision Date and Time XABs are input on close, so create an
                        09A3     ; XAB chain with these XABs if DAP Protection and/or Date and Time messages
                        09A3     ; have been received from partner.
                        09A3     ;
                        09A3
            04    AA    09A3             BICW2   #FAL$M_DATXAB,-                 ; Clear DATXAB flag as both DATXAB and
         72 A8          09A5                     FAL$W_RECEIVED(R8)              ;  RDTXABs are generated when a Date
                        09A7                                                     ;  and Time message is received
          0129    30    09A7             BSBW    CHAIN_RECV_XABS                 ; Add received XABs to XAB chain
            24    11    09AA             BRB     CLOSE_COMMON                    ; Join common close code
```

```
                    09AC          .SBTTL  FAL$CLOSE
                    09AC
                    09AC  ;++
                    09AC  ; This routine performs the DAP CLOSE file function.
                    09AC  ;--
                    09AC
                    09AC  FAL$CLOSE::                                     ; Entry point
         24 AA  D4  09AC          CLRL    FAB$L_XAB(R10)                  ; Remove any XABs from chain
         24 AA  DE  09AF          MOVAL   FAB$L_XAB(R10),-                ;
         7C A8      09B2                  FAL$L_CHAIN_NXT(R8)             ; Save address of next chain pointer
                    09B4
                    09B4  ;
                    09B4  ; Simulate support of the Revision Date and Time XAB on close if all of the
                    09B4  ; following are true:
                    09B4  ;   (1) partner does not support change of revision information on close
                    09B4  ;   (2) this is a file creation operation
                    09B4  ;   (3) a DAP Date and Time message was received on create
                    09B4  ;
                    09B4  ; If a Date and Time message is received on create, FAL$CREATE generates a
                    09B4  ; RDTXAB for use on create. This XAB can now be used as input on close to
                    09B4  ; preserve the revision date and time and revision number of the file that was
                    09B4  ; established on create. This is the desired action for a file COPY operation
                    09B4  ; where the remote node is not capable of specifying this action when the file
                    09B4  ; is closed.
                    09B4  ;
                    09B4
            2B  E0  09B4          BBS     #DAP$V_CHGTIMCLS,-              ; Branch if partner supports change
         28 A9      09B6                  DAP$Q_SYSCAP(R9),-              ;  of revision information on close
            17      09B8                  CLOSE_COMMON                    ;
       01F6 C8  91  09B9          CMPB    FAL$B_ACCFUNC(R8),-             ; Branch if this was not a file
            02      09BD                  #DAP$K_CREATE                   ;  creation operation
            10  12  09BE          BNEQ    CLOSE_COMMON                    ;
  57   03B0 C8  DE  09C0          MOVAL   FAL$L_RDTXAB(R8),R7             ; Get address of RDTXAB
    1E      67  91  09C5          CMPB    XAB$B_COD(R7),#XAB$C_RDT; Branch if RDTXAB has not been
            06  12  09C8          BNEQ    CLOSE_COMMON                    ;  initialized (and filled in on create)
          0178  30  09CA          BSBW    CHAIN_THIS_XAB                  ; Add this XAB to XAB chain
         04 A7  D4  09CD          CLRL    XAB$L_NXT(R7)                   ; Be sure XAB chain is terminated as
                    09D0                                                  ;  RDTXAB was on chain during create!
                    09D0
                    09D0  ;+
                    09D0  ; Common code path for FAL$RESET, FAL$CHANGE, and FAL$CLOSE.
                    09D0  ;-
                    09D0
                    09D0  CLOSE_COMMON:                                   ; Code common to all close operations
                    09D0
                    09D0  ;
                    09D0  ; Process optional DAP level CRC checksum value.
                    09D0  ;
                    09D0
            03  E1  09D0          BBC     #DAP$V_RET_CRC,-                ; Branch if checksum option not
    15 01F5 C8      09D2                  FAL$B_ACCOPT(R8),10$            ;  requested by accessing node
    11 69   01  E1  09D6          BBC     #DAP$V_X_CHECK,(R9),10$ ; Branch if checksum value not
                    09DA                                                  ;  explicitly sent by partner
         42 A9  B1  09DA          CMPW    DAP$W_CHECK(R9),-               ; Compare the checksums
         20 A9      09DD                  DAP$L_CRC_RSLT(R9)              ;  (order 16 CRC)
            0A  13  09DF          BEQLU   10$                             ; Branch if they match
         F61C'  30  09E1          BSBW    FAL$CRC_LOGERR                  ; Log the DAP CRC error
```

```
        50   82E2 8F   3C 09E4          MOVZWL  #<RMS$_CRC&^XFFFF>,R0   ; Stuff error code (lower 16 bits)
                  25   11 09E9          BRB     30$                     ; Return CRC error to partner
                          09EB
                          09EB     ;
                          09EB     ; Close the file.
                          09EB     ;
                          09EB
                          09EB     10$: $CLOSE  FAB=R10                 ; Close the file
               19 50   E9 09F4          BLBC    R0,30$                  ; Branch on failure
          51   04 AA   D0 09F7          MOVL    FAB$L_FOP(R10),R1       ; Get FOP options
               F602'   30 09FB          BSBW    FAL$LOG_CLSMSG          ; Log file close in print file
       03 68      0A   E0 09FE          BBS     #FAL$V_WILD,(R8),20$    ; Branch if wildcard operation
                016A   30 0A02          BSBW    SEND_CMP                ; Send Access Complete message
04 AA  0000E000 8F     CA 0A05     20$: BICL2   #<<FAB$M_DLT>!-         ; Clear FOP options (DLT, SCF, SPL)
                          0A0D                    <FAB$M_SCF>!-         ;  as wildcard file retrieval operation
                          0A0D                    <FAB$M_SPL>!-         ;  will not re-initialize the FAB and
                          0A0D                  0>,FAB$L_FOP(R10)       ;  these bits are input to RMS for
                          0A0D                                          ;  both open and close operations
                02D6   31 0A0D          BRW     EXIT_SUCCESS            ; Exit state with success
                028D   31 0A10     30$: BRW     ERR_FILE_CLOS           ; Return error in Status message
```

```
                 0A13          .SBTTL  STATE TABLE ERROR ROUTINES
                 0A13
                 0A13  ;++
                 0A13  ; The following routines are called by the state table manager to report state
                 0A13  ; table transition errors. These are due to an inappropriate DAP message being
                 0A13  ; received for the current state, or an invalid or unsupported operation being
                 0A13  ; requested for the current state. Each of these routines sends a Status message
                 0A13  ; to the partner process.
                 0A13  ;--
                 0A13
                 0A13
                 0A13          .SBTTL  FAL$OUT_OF_SEQ
                 0A13
                 0A13  ;++
                 0A13  ; The DAP message being processed is incorrect (out-of-sequence) for the state.
                 0A13  ;--
                 0A13
                 0A13  FAL$OUT_OF_SEQ::                              ; Entry point
     02AF   31   0A13          BRW     ERR_SYNCHRONIZE               ; Return error in Status message
                 0A16
                 0A16
                 0A16          .SBTTL  FAL$INV_ACCFUNC
                 0A16          .SBTTL  FAL$INV_CTLFUNC
                 0A16          .SBTTL  FAL$INV_CONFUNC, FAL$UNS_CONFUNC
                 0A16          .SBTTL  FAL$INV_CMPFUNC
                 0A16
                 0A16  ;++
                 0A16  ; The ACCFUNC field value is invalid for state table context.
                 0A16  ;--
                 0A16
                 0A16  FAL$INV_ACCFUNC::                             ; Entry point
50     10   D0   0A16          MOVL    #DAP$_ACCFUNC,R0              ; Get field ID code
     0297   31   0A19          BRW     ERR_INVALID                   ; Return error in Status message
                 0A1C
                 0A1C  ;++
                 0A1C  ; The CTLFUNC field value is invalid for state table context.
                 0A1C  ;--
                 0A1C
                 0A1C  FAL$INV_CTLFUNC::                             ; Entry point
50     10   D0   0A1C          MOVL    #DAP$_CTLFUNC,R0              ; Get field ID code
     0291   31   0A1F          BRW     ERR_INVALID                   ; Return error in Status message
                 0A22
                 0A22  ;++
                 0A22  ; The CONFUNC field value is invalid for state table context.
                 0A22  ;--
                 0A22
                 0A22  FAL$INV_CONFUNC::                             ; Entry point
50     10   D0   0A22          MOVL    #DAP$_CONFUNC,R0              ; Get field ID code
     028B   31   0A25          BRW     ERR_INVALID                   ; Return error in Status message
                 0A28
                 0A28  ;++
                 0A28  ; The ACCFUNC field value is unsupported for state table context.
                 0A28  ;--
                 0A28
                 0A28  FAL$UNS_ACCFUNC::                             ; Entry point
50     10   D0   0A28          MOVL    #DAP$_ACCFUNC,R0              ; Get field ID code
     028E   31   0A2B          BRW     ERR_UNSUPPORT                 ; Return error in Status message
```

```
                   0A2E
                   0A2E      ;++
                   0A2E      ; The CONFUNC field value is unsupported for state table context.
                   0A2E      ;--
                   0A2E
                   0A2E      FAL$UNS_CONFUNC::                              ; Entry point
     50    10  D0  0A2E              MOVL    #DAP$_CONFUNC,R0               ; Get field ID code
         0288  31  0A31              BRW     ERR_UNSUPPORT                  ; Return error in Status message
                   0A34
                   0A34      ;++
                   0A34      ; The CMPFUNC field value is invalid for state table context.
                   0A34      ;--
                   0A34
                   0A34      FAL$INV_CMPFUNC::                              ; Entry point
     50    10  D0  0A34              MOVL    #DAP$_CMPFUNC,R0               ; Get field ID code
         0279  31  0A37              BRW     ERR_INVALID                    ; Return error in Status message
                   0A3A
                   0A3A      ;++
                   0A3A      ; This routine returns a Status message with an unsupported-key-field error
                   0A3A      ; code. It is actually an internal support routine for FAL$DECODE_CTL.
                   0A3A      ;--
                   0A3A
                   0A3A      FAL$UNS_KEY::                                  ; Entry point
     50    13  D0  0A3A              MOVL    #DAP$_KEY,R0                   ; Get field ID code
         027C  31  0A3D              BRW     ERR_UNSUPPORT                  ; Return error in Status message
```

```
                          0A40          .SBTTL  SHARED SUPPORT ROUTINES
                          0A40
                          0A40
                          0A40          .SBTTL  CHAIN_RQST_XABS
                          0A40
                          0A40  ;++
                          0A40  ; This routine initializes XABs and adds them to the XAB chain as necessary
                          0A40  ; to obtain the requested file attribute information when the subsequent $OPEN
                          0A40  ; is performed.
                          0A40  ;
                          0A40  ; Note: Only requests for the DAP KEY, ALL, SUM, TIM, and PRO messages are
                          0A40  ;       handled here by chaining in the appropriate XABs. Furthermore, only
                          0A40  ;       primary Allocation and Key Definition XABs are chained into the list
                          0A40  ;       at this time, as the NOA and NOK values from the Summary XAB are not
                          0A40  ;       known yet.
                          0A40  ;
                          0A40  ; Inputs include:
                          0A40  ;
                          0A40  ;       FAL$W_DISPLAY
                          0A40  ;       FAL$L_CHAIN_NXT
                          0A40  ;--
                          0A40
                          0A40  CHAIN_RQST_XABS:                              ; Entry point
          5C    70 A8  3C 0A40          MOVZWL  FAL$W_DISPLAY(R8),AP        ; Copy display mask to scratch register
                          0A44                                              ;  (i.e., list of DAP Extended
                          0A44                                              ;  Attributes messages to return)
                74 A8  D4 0A44          CLRL    FAL$L_ALLXABINI(R8)         ; Zero list of ALLXABs initialized
                78 A8  D4 0A47          CLRL    FAL$L_KEYXABINI(R8)         ; Zero list of KEYXABs initialized
          5C    06     B3 0A4A          BITW    #<<DAP$M_DSP_ALL>!-         ; Branch if neither Allocation nor
                          0A4D                    <DAP$M_DSP_KEY>!-         ;  Key Definition messages are to be
                          0A4D                  0>,AP                       ;  returned
                04     13 0A4D          BEQL    10$                         ;
                          0A4F          $SETBIT #DAP$V_DSP_SUM,AP           ; Include Summary XAB (to obtain NOA and
                          0A53                                              ;  NOK values) in case an indexed file
                          0A53                                              ;  is opened
                56     D4 0A53  10$:    CLRL    R6                          ; Indicate AID=0 and REF=0 for
                          0A55                                              ;  Allocation and Key Definition XABs
                          0A55
                          0A55          ASSUME  DAP$V_DSP_ATT+1 EQ DAP$V_DSP_KEY
                          0A55          ASSUME  DAP$V_DSP_KEY+1 EQ DAP$V_DSP_ALL
                          0A55          ASSUME  DAP$V_DSP_ALL+1 EQ DAP$V_DSP_SUM
                          0A55          ASSUME  DAP$V_DSP_SUM+1 EQ DAP$V_DSP_TIM
                          0A55          ASSUME  DAP$V_DSP_TIM+1 EQ DAP$V_DSP_PRO
                          0A55
                          0A55  CHAIN_RQST_LOOP:                              ;
    50    5C    05  01 EA 0A55          FFS     #DAP$V_DSP_KEY,#5,AP,R0     ; Get position of next bit set
                          0A5A          $CLRBIT R0,AP                       ; Clear request bit just found
              72'AF    9F 0A5E          PUSHAB  B^10$                       ; Push return address on stack
                          0A61          $CASEB  SELECTOR=R0-                ; Next XAB to add to chain:
                          0A61                  BASE=#DAP$V_DSP_KEY-        ;
                          0A61                  DISPL=<-                    ;
                          0A61                      FAL$INIT_KEYXAB-        ;  Key Definition
                          0A61                      FAL$INIT_ALLXAB-        ;  Allocation
                          0A61                      FAL$INIT_SUMXAB-        ;  Summary
                          0A61                      FAL$INIT_DATXAB-        ;  Date and Time
                          0A61                      FAL$INIT_PROXAB-        ;  Protection
                          0A61                  >                           ;
```

```
        01   BA  0A6F  2121        POPR    #^M<R0>              ; Throw away loop return address
             05  0A71  2122        RSB                          ; Exit
      00D0   30  0A72  2123 10$:   BSBW    CHAIN_THIS_XAB       ; Add this XAB to XAB chain
        DE   11  0A75  2124        BRB     CHAIN_RQST_LOOP      ; Handle next request
```

```
                         0A77        .SBTTL  CHAIN_RQST_PRT2
                         0A77
                         0A77  ;++
                         0A77  ; This routine creates an XAB chain of alternate Allocation and Key Definition
                         0A77  ; XABs (as required) and then calls $DISPLAY to fill them in.
                         0A77  ;
                         0A77  ; If an indexed file was opened and DAP Allocation and/or Key Definition
                         0A77  ; messages are to be returned, then the NOA and NOK values from the Summary XAB
                         0A77  ; are used to determine how may alternate XABs are required to be filled.
                         0A77  ;--
                         0A77
                         0A77  CHAIN_RQST_PRT2:                                    ; Entry point
         50    01    D0  0A77          MOVL    #1,R0                               ; Assume success
         1D AA       91  0A7A          CMPB    FAB$B_ORG(R10),-                    ; Branch if not IDX organization
            20           0A7D                  #FAB$C_IDX                          ;
            52       12  0A7E          BNEQ    50$                                 ;
      70 A8    06    B3  0A80          BITW    #<<DAP$M_DSP_ALL>!-                 ; Branch if neither Allocation nor
                         0A84                    <DAP$M_DSP_KEY>!-                 ;  Key Definition messages are to be
                         0A84                  0>,FAL$W_DISPLAY(R8)                ;  returned
            4C       13  0A84          BEQL    50$                                 ;
         24 AA       DE  0A86          MOVAL   FAB$L_XAB(R10),-                    ; Get address of start of XAB chain
         7C A8           0A89                  FAL$L_CHAIN_NXT(R8)                 ;  and store it
                         0A8B
                         0A8B  ;
                         0A8B  ; Initialize multiple Allocation XABs as required and chain them in.
                         0A8B  ;
                         0A8B
            02       E1  0A8B          BBC     #DAP$V_DSP_ALL,-                    ; Branch if DAP Allocation message
      1A 70 A8           0A8D                  FAL$W_DISPLAY(R8),20$               ;  is not requested
 5C   03AC C8        9A  0A90          MOVZBL  FAL$L_SUMXAB+XAB$B_NOA(R8),AP ; Get number of areas value
   01       5C       D1  0A95          CMPL    AP,#1                               ; Branch if there are no alternate
            10       1B  0A98          BLEQU   20$                                 ;  areas
            56       D4  0A9A          CLRL    R6                                  ; Initialize area ID value
        F561'        30  0A9C  10$:    BSBW    FAL$INIT_ALLXAB                     ; Initialize next Allocation XAB
                         0A9F                                                      ;  on return R7 = address of XAB
         08 50       E9  0A9F          BLBC    R0,20$                              ; Branch on failure (AID too large)
          00A0       30  0AA2          BSBW    CHAIN_THIS_XAB                      ; Add this XAB to XAB chain
            56       D6  0AA5          INCL    R6                                  ; Increment area ID value
         F2 5C       F5  0AA7          SOBGTR  AP,10$                              ; Branch if more to process
                         0AAA
                         0AAA  ;
                         0AAA  ; Initialize multiple Key Definition XABs (as required) and chain them in.
                         0AAA  ;
                         0AAA
            01       E1  0AAA  20$:    BBC     #DAP$V_DSP_KEY,-                    ; Branch if DAP Key Definition message
      1A 70 A8           0AAC                  FAL$W_DISPLAY(R8),40$               ;  is not requested
 5C   03AD C8        9A  0AAF          MOVZBL  FAL$L_SUMXAB+XAB$B_NOK(R8),AP ; Get number of keys value
   01       5C       D1  0AB4          CMPL    AP,#1                               ; Branch if there are no alternate
            10       1B  0AB7          BLEQU   40$                                 ;  keys
            56       D4  0AB9          CLRL    R6                                  ; Initialize key of reference value
        F542'        30  0ABB  30$:    BSBW    FAL$INIT_KEYXAB                     ; Initialize next Key Definition XAB
                         0ABE                                                      ;  on return R7 = address of XAB
         08 50       E9  0ABE          BLBC    R0,40$                              ; Branch on failure (REF too large)
          0081       30  0AC1          BSBW    CHAIN_THIS_XAB                      ; Add this XAB to XAB chain
            56       D6  0AC4          INCL    R6                                  ; Increment key of reference value
         F2 5C       F5  0AC6          SOBGTR  AP,30$                              ; Branch if more to process
                         0AC9
```

```
         0AC9  ;
         0AC9  ; Obtain additional file attribute information.
         0AC9  ;
         0AC9
         0AC9  40$:     $DISPLAY FAB=R10                    ; Fill in primary and alternate
         0AD2                                               ;  Allocation and Key Definition XABs
   05    0AD2  50$:     RSB                                 ; Exit
```

```
                         0AD3          .SBTTL  CHAIN_RECV_XABS
                         0AD3
                         0AD3  ;++
                         0AD3  ; This routine adds XABs to the XAB chain that were received from the partner
                         0AD3  ; process for use when a subsequent $CREATE, $EXTEND, or $CLOSE is performed.
                         0AD3  ;
                         0AD3  ; Note: Only XABs generated from the DAP KEY, ALL, TIM, and PRO messages are
                         0AD3  ;       handled here by chaining in the appropriate XABs. However, in contrast
                         0AD3  ;       To CHAIN_RQST_XABs, multiple Allocation and Key Definition XABs are
                         0AD3  ;       chained into the list (if present).
                         0AD3  ;
                         0AD3  ; Inputs include:
                         0AD3  ;
                         0AD3  ;       FAL$W_RECEIVED
                         0AD3  ;       FAL$L_KEYXABINI
                         0AD3  ;       FAL$L_ALLXABINI
                         0AD3  ;       FAL$L_CHAIN_NXT
                         0AD3  ;--
                         0AD3
                         0AD3  CHAIN_RECV_XABS:                             ; Entry point
         5C   72 A8   3C 0AD3          MOVZWL  FAL$W_RECEIVED(R8),AP        ; Copy received mask to scratch register
                         0AD7                                               ;  (i.e., list of XABs generated from
                         0AD7                                               ;  receipt of optional DAP messages)
                         0AD7
                         0AD7          ASSUME  FAL$V_KEYXAB    EQ 0
                         0AD7          ASSUME  FAL$V_KEYXAB+1 EQ FAL$V_ALLXAB
                         0AD7          ASSUME  FAL$V_ALLXAB+1 EQ FAL$V_DATXAB
                         0AD7          ASSUME  FAL$V_DATXAB+1 EQ FAL$V_PROXAB
                         0AD7          ASSUME  FAL$V_PROXAB+1 EQ FAL$V_RDTXAB
                         0AD7
                         0AD7  CHAIN_RECV_LOOP:                             ;
   50   5C   05   00  EA 0AD7          FFS     #FAL$V_KEYXAB,#5,AP,R0       ; Get position of next bit set
                         0ADC          $CLRBIT R0,AP                        ; Clear request bit just found
                         0AE0          $CASEB  SELECTOR=R0-                 ; Next XAB to add to chain:
                         0AE0                  DISPL=<-                     ;
                         0AE0                          10$-                 ;  Key definition
                         0AE0                          20$-                 ;  Allocation
                         0AE0                          30$-                 ;  Date and Time
                         0AE0                          40$-                 ;  Protection
                         0AE0                          50$-                 ;  Revision Date and Time
                         0AE0                  >                            ;
                      05 0AEE          RSB                                  ; Exit
                         0AEF
                         0AEF  ;
                         0AEF  ; Chain Key Definition XAB(s).
                         0AEF  ;
                         0AEF
         55   78 A8   D0 0AEF  10$:    MOVL    FAL$L_KEYXABINI(R8),R5       ; Put list of KEYXABs received in
                         0AF3                                               ;  scratch register; at least one bit
                         0AF3                                               ;  in range is guaranteed to be set
   56   55   20   00  EA 0AF3  15$:    FFS     #0,#FAL$K_MAX_REF+1,R5,R6;Get REF value of next KEYXAB
                  DD  13 0AF8          BEQL    CHAIN_RECV_LOOP              ; Branch if mask exhausted
                         0AFA          $CLRBIT R6,R5                        ; Clear bit just found
   56  0000004C 8F    C4 0AFE          MULL2   #FAL$K_KEYXAB,R6             ; Using REF as an index, compute
      57   1000 C846  9E 0B05          MOVAB   FAL$L_KEYXAB(R8)[R6],R7      ;  address of KEYXAB to use
            0037      30 0B0B          BSBW    CHAIN_THIS_XAB               ; Add this XAB to XAB chain
              E3      11 0B0E          BRB     15$                          ; Handle next alternate KEYXAB
```

```
                                 0B10
                                 0B10      ;
                                 0B10      ; Chain Allocation XAB(s).
                                 0B10      ;
                                 0B10
            55    74 A8    D0    0B10      20$:    MOVL    FAL$L_ALLXABINI(R8),R5     ; Put list of ALLXABs received in
                                 0B14                                                 ;  scratch register; at least one bit
                                 0B14                                                 ;  in range is guaranteed to be set
     56     55    20    00    EA 0B14      25$:    FFS     #0,#FAL$K_MAX_AID+1,R5,R6;Get AID value of next ALLXAB
                        BC    13 0B19              BEQL    CHAIN_RECV_LOOP            ; Branch if mask exhausted
                                 0B1B              $CLRBIT R6,R5                      ; Clear bit just found
                  56    20    C4 0B1F              MULL2   #FAL$K_ALLXAB,R6           ; Using AID as an index, compute
            57  0C00 C846     9E 0B22              MOVAB   FAL$L_ALLXAB(R8)[R6],R7    ;  address of ALLXAB to use
                      001A    30 0B28              BSBW    CHAIN_THIS_XAB             ; Add this XAB to XAB chain
                        E7    11 0B2B              BRB     25$                        ; Handle next alternate ALLXAB
                                 0B2D
                                 0B2D      ;
                                 0B2D      ; Chain Date and Time, Protection, and Revision Date and Time XABs.
                                 0B2D      ;
                                 0B2D
            57  0320 C8   DE     0B2D      30$:    MOVAL   FAL$L_DATXAB(R8),R7        ; Get address of Date and Time XAB
                      0C   11    0B32              BRB     60$                        ;
            57  034C C8   DE     0B34      40$:    MOVAL   FAL$L_PROXAB(R8),R7        ; Get address of Protection XAB
                      05   11    0B39              BRB     60$                        ;
            57  03B0 C8   DE     0B3B      50$:    MOVAL   FAL$L_RDTXAB(R8),R7        ; Get address of Rev Date and Time XAB
                    0002   30    0B40      60$:    BSBW    CHAIN_THIS_XAB             ; Add this XAB to XAB chain
                      92   11    0B43              BRB     CHAIN_RECV_LOOP            ; Handle next request
```

```
                      0B45          .SBTTL  CHAIN_THIS_XAB
                      0B45
                      0B45   ;++
                      0B45   ; This routine chains the specified XAB into the XAB chain.
                      0B45   ;
                      0B45   ; Inputs include:
                      0B45   ;
                      0B45   ;       R7 contains address of XAB
                      0B45   ;
                      0B45   ;       FAL$L_CHAIN_NXT
                      0B45   ;
                      0B45   ;--
                      0B45
                      0B45   CHAIN_THIS_XAB:                                ; Entry point
    7C B8    57   D0  0B45          MOVL    R7,@FAL$L_CHAIN_NXT(R8)  ; Insert XAB address into XAB chain
       04    57   C1  0B49          ADDL3   R7,#XAB$L_NXT,-          ; Compute address of next chain pointer
          7C A8       0B4C                  FAL$L_CHAIN_NXT(R8)      ;  and save it
                  05  0B4E          RSB                              ; Exit
                      0B4F
                      0B4F
                      0B4F          .SBTTL  CHECK_FOR_PMR
                      0B4F
                      0B4F   ;++
                      0B4F   ; This routine is intended to be called after an RMS file parse operation has
                      0B4F   ; been performed. It checks for the existence of a node name in the file
                      0B4F   ; specification, and if found, it returns an error if the FAL logging option
                      0B4F   ; to disable poor-man's routing (also called manual routing) has been selected.
                      0B4F   ;
                      0B4F   ; Note: Poor-man's routing is an unsupported and undocumented feature.
                      0B4F   ;--
                      0B4F
                      0B4F   CHECK_FOR_PMR:                                 ; Entry point
       0F 50      E9  0B4F          BLBC    R0,10$                   ; Do not alter error status from
                      0B52                                           ;  previous RMS parse operation
          11      E1  0B52          BBC     #NAM$V_NODE,-            ; Exit if file specification did not
     02C8 C8          0B54                  FAL$L_NAM+NAM$L_FNB(R8),-; yield a node name (i.e., initiator
          09          0B57                  10$                      ;  did not specify multiple node names)
    05 68    33   E1  0B58          BBC     #FAL$V_DIS_PMR,(R8),10$  ; Allow FAL to act as file access router
                      0B5C                                           ;  unless feature is disabled by user
50    85F4 8F     3C  0B5C          MOVZWL  #<RMS$_NOD&^XFFFF>,R0    ; Generate 'invalid node name' error
                  05  0B61   10$:   RSB                              ; Exit
```

```
                  0B62          .SBTTL  SEND_ACK
                  0B62
                  0B62  ;++
                  0B62  ; Build and send Acknowledge message to partner.
                  0B62  ;--
                  0B62
                  0B62  SEND_ACK:                                     ; Entry point
    50    06  D0  0B62          MOVL    #DAP$K_ACK_MSG,R0             ; Get message type value
       F498'  30  0B65          BSBW    FAL$BUILD_HEAD                ; Construct message header
                  0B68                                                ; There are no fields in message body!
       F495'  30  0B68          BSBW    FAL$BUILD_TAIL                ; Finish building message
       F492'  30  0B6B          BSBW    FAL$TRANSMIT                  ; Send Acknowledge message
              05  0B6E          RSB                                   ; Return
                  0B6F
                  0B6F
                  0B6F          .SBTTL  SEND_CMP
                  0B6F
                  0B6F  ;++
                  0B6F  ; Build and send Access Complete message to partner.
                  0B6F  ; Declare this last message of a response sequence and exit state with success.
                  0B6F  ;--
                  0B6F
                  0B6F  SEND_CMP:                                     ; Entry point
                  0B6F          $SETBIT #FAL$V_LAST_MSG,(R8)          ; Declare this last message to block
    50    07  D0  0B73          MOVL    #DAP$K_CMP_MSG,R0             ; Get message type value
       F487'  30  0B76          BSBW    FAL$BUILD_HEAD                ; Construct message header
    83    02  90  0B79          MOVB    #DAP$K_RESPONSE,(R3)+         ; Store CMPFUNC field
       F481'  30  0B7C          BSBW    FAL$BUILD_TAIL                ; Finish building message
       F47E'  30  0B7F          BSBW    FAL$TRANSMIT                  ; Send Access Complete message
              05  0B82          RSB                                   ; Exit
```

```
                        0B83         .SBTTL  SEND_OPTNL_MSGS
                        0B83
                        0B83   ;++
                        0B83   ; Build and send (main) Attributes, Extended Attributes, and Name messages to
                        0B83   ; partner as directed by the request mask (FAL$W_DISPLAY).
                        0B83   ;
                        0B83   ; Note: The Summary message will be sent before any of the other Extended
                        0B83   ;       Attributes messages (per DAP V6.0 specification).
                        0B83   ;
                        0B83   ; Inputs include:
                        0B83   ;
                        0B83   ;       FAL$W_DISPLAY
                        0B83   ;       FAL$L_KEYXABINI
                        0B83   ;       FAL$L_ALLXABINI
                        0B83   ;--
                        0B83
                        0B83   SEND_OPTNL_MSGS:                                  ; Entry point
         5C  70 A8  3C  0B83         MOVZWL  FAL$W_DISPLAY(R8),AP       ; Copy display mask to scratch register
                        0B87                                            ;  (i.e., list of DAP Extended
                        0B87                                            ;  Attributes messages to return)
                        0B87
                        0B87         ASSUME  DAP$V_DSP_ATT    EQ 0
                        0B87         ASSUME  DAP$V_DSP_ATT+1 EQ DAP$V_DSP_KEY
                        0B87         ASSUME  DAP$V_DSP_KEY+1 EQ DAP$V_DSP_ALL
                        0B87         ASSUME  DAP$V_DSP_ALL+1 EQ DAP$V_DSP_SUM
                        0B87         ASSUME  DAP$V_DSP_SUM+1 EQ DAP$V_DSP_TIM
                        0B87         ASSUME  DAP$V_DSP_TIM+1 EQ DAP$V_DSP_PRO
                        0B87         ASSUME  DAP$V_DSP_PRO+3 EQ DAP$V_DSP_NAM
                        0B87
50  5C   02   01    EF  0B87         EXTZV   #DAP$V_DSP_KEY,#2,AP,R0 ; Rearrange display mask to put
51  5C   01   03    EF  0B8C         EXTZV   #DAP$V_DSP_SUM,#1,AP,R1 ;  Summary message ahead of
5C  01   01   51    F0  0B91         INSV    R1,#DAP$V_DSP_KEY,#1,AP ;  Key Definition and
5C  02   02   50    F0  0B96         INSV    R0,#DAP$V_DSP_ALL,#2,AP ;  Allocation messages
                        0B9B   SEND_OPTNL_LOOP:
50  5C   09   00    EA  0B9B         FFS     #0,#DAP$V_DSP_NAM+1,AP,R0;Get position of next bit set
                        0BA0         $CLRBIT R0,AP                      ; Clear request bit just found
            11'AF   9F  0BA4         PUSHAB  B^50$                      ; Push return address on stack
                        0BA7         $CASEB  SELECTOR=R0-               ; Next message:
                        0BA7                 DISPL=<-                   ;
                        0BA7                         FAL$ENCODE_ATT-    ;  Attributes
                        0BA7                         FAL$ENCODE_SUM-    ;  Summary
                        0BA7                         10$-               ;  Key Definition
                        0BA7                         20$-               ;  Allocation
                        0BA7                         FAL$ENCODE_TIM-    ;  Date and Time
                        0BA7                         FAL$ENCODE_PRO-    ;  Protection
                        0BA7                         40$-               ;  (reserved)
                        0BA7                         40$-               ;  (reserved)
                        0BA7                         FAL$ENCODE_NAM-    ;  (resultant) Name
                        0BA7                 >                          ; Exit loop
               01   BA  0BBD         POPR    #^M<R0>                    ; Throw away loop return address
               05       0BBF         RSB                                ; Exit
               5C   DD  0BC0   10$:  PUSHL   AP                         ; Save register
         5C  78 A8  D0  0BC2         MOVL    FAL$L_KEYXABINI(R8),AP     ; Put list of KEYXABs initialized in
                        0BC6                                            ;  scratch register; at least one bit
                        0BC6                                            ;  is guaranteed to be set
56  5C   20   00    EA  0BC6   15$:  FFS     #0,#FAL$K_MAX_REF+1,AP,R6;Get REF value of next KEYXAB
               3D   13  0BCB         BEQL    30$                        ; Branch if mask exhausted
```

```
                              0BCD  2405          $CLRBIT R6,AP                         ; Clear bit just found
    50  0000004C 8F   56   C5 0BD1  2406          MULL3   R6,#FAL$K_KEYXAB,R0           ; Using REF as an index, compute
          57   1000 C840   9E 0BD9  2407          MOVAB   FAL$L_KEYXAB(R8)[R0],R7       ;  address of KEYXAB to use
                  F41E'   30 0BDF  2408          BSBW    FAL$ENCODE_KEY                ; Build next Key Definition message
                  F41B'   30 0BE2  2409          BSBW    FAL$TRANSMIT                  ; Send message just built
                     DF   11 0BE5  2410          BRB     15$                           ; Handle next alternate KEYXAB
                     5C   DD 0BE7  2411 20$:     PUSHL   AP                            ; Save register
          5C     74 A8   D0 0BE9  2412          MOVL    FAL$L_ALLXABINI(R8),AP        ; Put list of ALLXABs initialized in
                              0BED  2413                                                ;  scratch register; at least one bit
                              0BED  2414                                                ;  in range is guaranteed to be set
    56   5C   20   00     EA 0BED  2415 25$:     FFS     #0,#FAL$K_MAX_AID+1,AP,R6;Get AID value of next ALLXAB
                     16   13 0BF2  2416          BEQL    30$                           ; Branch if mask exhausted
                              0BF4  2417          $CLRBIT R6,AP                         ; Clear bit just found
         50   20   56    C5 0BF8  2418          MULL3   R6,#FAL$K_ALLXAB,R0           ; Using AID as an index, compute
          57   0C00 C840   9E 0BFC  2419          MOVAB   FAL$L_ALLXAB(R8)[R0],R7       ;  address of ALLXAB to use
                  F3FB'   30 0C02  2420          BSBW    FAL$ENCODE_ALL                ; Build next Allocation message
                  F3F8'   30 0C05  2421          BSBW    FAL$TRANSMIT                  ; Send message just built
                     E3   11 0C08  2422          BRB     25$                           ; Handle next alternate ALLXAB
                  5C 8ED0    0C0A  2423 30$:     POPL    AP                            ; Restore register
                     01   BA 0C0D  2424 40$:     POPR    #^M<R0>                       ; Throw away loop return address
                     8A   11 0C0F  2425          BRB     SEND_OPTNL_LOOP               ; Handle next request
                  F3EC'   30 0C11  2426 50$:     BSBW    FAL$TRANSMIT                  ; Send message just built
                     85   11 0C14  2427          BRB     SEND_OPTNL_LOOP               ; Loop until done
```

```
                         0C16          .SBTTL  SEND_3PART_NAM
                         0C16
                         0C16  ;++
                         0C16  ; Build and send the (volume) Name, (directory) Name, and (file) Name messages
                         0C16  ; to partner.
                         0C16  ;
                         0C16  ; On output R1-R7 and AP are destroyed.
                         0C16  ;--
                         0C16
                         0C16  SEND_3PART_NAM:                             ; Entry point
                01   BB  0C16          PUSHR   #^M<R0>                     ; Save status code
          5C  28 AA  D0  0C18          MOVL    FAB$L_NAM(R10),AP           ; Get address of NAM block
                         0C1C
                         0C1C  ;+
                         0C1C  ; Return DAP (volume) Name message to partner.
                         0C1C  ; The volume string in this context means nodespec (if present) plus device name
                         0C1C  ; string.
                         0C1C  ;-
                         0C1C
                         0C1C  SEND_VOL_NAM:                               ;
     54    0080 C8   7D  0C1C          MOVQ    FAL$Q_VOLNAME(R8),R4        ; <R4,R5> => previous volume name
      56     38 AC   9A  0C21          MOVZBL  NAM$B_NODE(AP),R6           ; Get size of node name string
      56     39 AC   80  0C25          ADDB2   NAM$B_DEV(AP),R6            ; Add size of device name string
      57     40 AC   D0  0C29          MOVL    NAM$L_NODE(AP),R7           ; <R6,R7> => current volume name
67   56  00  65  54  2D  0C2D          CMPC5   R4,(R5),#0,R6,(R7)          ; Branch if previous and current
                 16  13  0C33          BEQL    SEND_DIR_NAM                ;  strings match
       0080 C8   56  D0  0C35          MOVL    R6,FAL$Q_VOLNAME(R8)        ; It's a new one, so save it
       65   67   56  28  0C3A          MOVC3   R6,(R7),(R5)                ;  (making it the previous one)
            55   08  9A  0C3E          MOVZBL  #DAP$M_VOLNAME,R5           ; Get name type parameter
             F3BC'   30  0C41          BSBW    FAL$ENCODE_NAM1             ; Build (volume) Name message
             F3B9'   30  0C44          BSBW    FAL$TRANSMIT                ; Send Name message
       0088 C8       D4  0C47          CLRL    FAL$Q_DIRNAME(R8)           ; Clear the directory spec to force
                         0C4B                                              ; sending of directory name msg.
                         0C4B
                         0C4B  ;+
                         0C4B  ; Return DAP (directory) Name message to partner.
                         0C4B  ;-
                         0C4B
                         0C4B  SEND_DIR_NAM:                               ;
     54    0088 C8   7D  0C4B          MOVQ    FAL$Q_DIRNAME(R8),R4        ; <R4,R5> => previous directory name
      56     3A AC   9A  0C50          MOVZBL  NAM$B_DIR(AP),R6            ; Get size of directory name string
      57     48 AC   D0  0C54          MOVL    NAM$L_DIR(AP),R7            ; <R6,R7> => current directory name
67   56  00  65  54  2D  0C58          CMPC5   R4,(R5),#0,R6,(R7)          ; Branch if previous and current
                 12  13  0C5E          BEQL    SEND_FIL_NAM                ;  strings match
       0088 C8   56  D0  0C60          MOVL    R6,FAL$Q_DIRNAME(R8)        ; It's a new one, so save it
       65   67   56  28  0C65          MOVC3   R6,(R7),(R5)                ;  (making it the previous one)
            55   04  9A  0C69          MOVZBL  #DAP$M_DIRNAME,R5           ; Get name type parameter
             F391'   30  0C6C          BSBW    FAL$ENCODE_NAM1             ; Build (directory) Name message
             F38E'   30  0C6F          BSBW    FAL$TRANSMIT                ; Send Name message
                         0C72
                         0C72  ;+
                         0C72  ; Return DAP (file) Name message to partner.
                         0C72  ; The file string in this context can be either:
                         0C72  ;   (1) file name plus file type plus file version strings concatenated,
                         0C72  ;       which is the normal case, or
                         0C72  ;   (2) quoted string (if FAL received a file specification of the form
                         0C72  ;       nodespec::"foreign-filespec").
```

```
                    0C72  ;-
                    0C72
                    0C72
                    0C72  SEND_FIL_NAM:                                    ;
56     3B AC   9A   0C72        MOVZBL  NAM$B_NAME(AP),R6                  ; Get size of file name string
56     3C AC   80   0C76        ADDB2   NAM$B_TYPE(AP),R6                  ; Add size of file type string
56     3D AC   80   0C7A        ADDB2   NAM$B_VER(AP),R6                   ; Add size of file version string
57     4C AC   D0   0C7E        MOVL    NAM$L_NAME(AP),R7                  ; <R6,R7> => current file name
    55    02   9A   0C82        MOVZBL  #DAP$M_FILNAME,R5                  ; Get name type parameter
       F378'   30   0C85        BSBW    FAL$ENCODE_NAM1                    ; Build (file) Name message
       F375'   30   0C88        BSBW    FAL$TRANSMIT                       ; Send Name message
          01   BA   0C8B        POPR    #^M<R0>                            ; Restore status code
               05   0C8D        RSB                                        ; Exit
```

```
                              0C8E         .SBTTL  SHARED STATE EXIT ROUTINES
                              0C8E
                              0C8E   ;++
                              0C8E   ; Return DAP file open error.
                              0C8E   ; R0 contains RMS completion code on input.
                              0C8E   ;--
                              0C8E
                              0C8E   ERR_FILE_OPEN:                                  ; Entry point
          52    04      D0    0C8E           MOVL    #DAP$_FILE_OPEN,R2              ; Get DAP MACCODE value
    51       0C AA      D0    0C91           MOVL    FAB$L_STV(R10),R1               ; Get status value from FAB
                10      11    0C95           BRB     ERR_FILE                        ; Send Status message
                              0C97
                              0C97   ;++
                              0C97   ; Return DAP file transfer error.
                              0C97   ; R0 contains RMS completion code on input.
                              0C97   ;--
                              0C97
                              0C97   ERR_FILE_XFER:                                  ; Entry point
          52    05      D0    0C97           MOVL    #DAP$_FILE_XFER,R2              ; Get DAP MACCODE value
    51       0C AB      D0    0C9A           MOVL    RAB$L_STV(R11),R1               ; Get status value from RAB
                07      11    0C9E           BRB     ERR_FILE                        ; Send Status message
                              0CA0
                              0CA0   ;++
                              0CA0   ; Return DAP file close error.
                              0CA0   ; R0 contains RMS completion code on input.
                              0CA0   ;--
                              0CA0
                              0CA0   ERR_FILE_CLOS:                                  ; Entry point
          52    07      D0    0CA0           MOVL    #DAP$_FILE_CLOS,R2              ; Get DAP MACCODE value
    51       0C AA      D0    0CA3           MOVL    FAB$L_STV(R10),R1               ; Get status value from FAB
                              0CA7   ERR_FILE:                                       ;
                08      13    0CA7           BEQL    10$                             ; Branch on zero STV
    04 69       34      E1    0CA9           BBC     #DAP$V_VAXVMS,(R9),10$          ; Branch if partner is not VAX/VMS
                              0CAD           $SETBIT #FAL$V_RET_STV,(R8)             ; Request return of STV field in
                              0CB1                                                   ;  DAP Status message
                19      11    0CB1   10$:    BRB     EXIT_STS_FAIL                   ; Send Status message
                              0CB3
                              0CB3   ;++
                              0CB3   ; Return DAP invalid error.
                              0CB3   ; R0 contains DAP field ID code on input.
                              0CB3   ;--
                              0CB3
                              0CB3   ERR_INVALID:                                    ; Entry point
    51       30 A9      9A    0CB3           MOVZBL  DAP$B_TYPE(R9),R1               ; Get DAP message type
          52    09      D0    0CB7           MOVL    #DAP$_INVALID,R2                ; Get DAP MACCODE value
                10      11    0CBA           BRB     EXIT_STS_FAIL                   ; Send Status message
                              0CBC
                              0CBC   ;++
                              0CBC   ; Return DAP unsupported error.
                              0CBC   ; R0 contains DAP field ID code on input.
                              0CBC   ;--
                              0CBC
                              0CBC   ERR_UNSUPPORT:                                  ; Entry point
    51       30 A9      9A    0CBC           MOVZBL  DAP$B_TYPE(R9),R1               ; Get DAP message type
          52    02      D0    0CC0           MOVL    #DAP$_UNSUPPORT,R2              ; Get DAP MACCODE value
                07      11    0CC3           BRB     EXIT_STS_FAIL                   ; Send Status message
                              0CC5
```

```
                     0CC5  ;++
                     0CC5  ; Return DAP message-out-of-sequence error.
                     0CC5  ;--
                     0CC5
                     0CC5  ERR_SYNCHRONIZE:                          ; Entry point
 51   30 A9     9A   0CC5          MOVZBL  DAP$B_TYPE(R9),R1         ; Get DAP message type
    52   0A     D0   0CC9          MOVL    #DAP$_MSG_SYNC,R2         ; Get DAP MACCODE value
                     0CCC
                     0CCC  ;++
                     0CCC  ; Build and send (failure) Status message to partner.
                     0CCC  ; Declare this last message of a response sequence and exit state with failure.
                     0CCC  ; R0 contains RMS completion code on input.
                     0CCC  ;--
                     0CCC
                     0CCC  EXIT_STS_FAIL:                            ; Entry point
                     0CCC          $SETBIT #FAL$V_LAST_MSG,(R8)      ; Declare this last message to block
      F32D'     30   0CD0          BSBW    FAL$ENCODE_STS            ; Build Status message
      F32A'     30   0CD3          BSBW    FAL$TRANSMIT              ; Send Status message to partner
                     0CD6
                     0CD6  ;++
                     0CD6  ; Exit state with failure.
                     0CD6  ;--
                     0CD6
                     0CD6  EXIT_FAILURE:                             ; Entry point
         50     D4   0CD6          CLRL    R0                        ; Signal state transition failure
                05   0CD8          RSB                               ; Exit to state table manager
                     0CD9
                     0CD9  ;++
                     0CD9  ; Build and send (success) Status message to partner.
                     0CD9  ; Declare this last message of a response sequence and exit state with success.
                     0CD9  ; R0 contains RMS (success) completion code on input.
                     0CD9  ;--
                     0CD9
                     0CD9  EXIT_STS_SUC:                             ; Entry point
                     0CD9          $SETBIT #FAL$V_LAST_MSG,(R8)      ; Declare this last message to block
    52   01     D0   0CDD          MOVL    #DAP$_SUCCESS,R2          ; Get DAP MACCODE value
      F31D'     30   0CE0          BSBW    FAL$ENCODE_STS            ; Build Status message
      F31A'     30   0CE3          BSBW    FAL$TRANSMIT              ; Send Status message to partner
                     0CE6
                     0CE6  ;++
                     0CE6  ; Exit state with success.
                     0CE6  ;--
                     0CE6
                     0CE6  EXIT_SUCCESS:                             ; Entry point
    50   01     D0   0CE6          MOVL    #1,R0                     ; Signal state transition success
                05   0CE9          RSB                               ; Exit to state table manager
                     0CEA
                     0CEA          .END                              ; End of module
```

```
$$.TMP1                  = 00000001               DAP$B_NUL                  0000006D
$$.TMP2                  = 0000005A               DAP$B_ORG                  00000045
$$COUNT                  = 00000009               DAP$B_OSTYPE               00000042
CHAIN_RECV_LOOP            00000AD7 R     03      DAP$B_RAC                  00000046
CHAIN_RECV_XABS            00000AD3 R     03      DAP$B_RAT                  00000047
CHAIN_RQST_LOOP            00000A55 R     03      DAP$B_REF                  0000006C
CHAIN_RQST_PRT2            00000A77 R     03      DAP$B_RFM                  00000046
CHAIN_RQST_XABS            00000A40 R     03      DAP$B_SHR                  00000043
CHAIN_THIS_XAB             00000B45 R     03      DAP$B_SIZ                  0000005C
CHECK_FOR_PMR              00000B4F R     03      DAP$B_SIZ_TMP              0000004A
CHECK_PARSE                0000000A R     03      DAP$B_STREAMID             00000032
CHECK_STREAM               00000473 R     03      DAP$B_TKS                  0000007F
CLOSE_COMMON               000009D0 R     03      DAP$B_TYPE                 00000030
COMPUTE_RCV_CRC            00000685 R     03      DAP$B_USRNUM               00000046
COMPUTE_XMT_CRC            00000454 R     03      DAP$B_USRVER               00000048
CR                       = 0000000D               DAP$B_VERNUM               00000044
CRLF                     = 00000A0D               DAP$B_X_FIELD              00000024
CTRLZ                    = 0000001A               DAP$C_BLN                  000000C0
DAP$B_ACCFUNC              00000040               DAP$K_ACK_MSG            = 00000006
DAP$B_ACCOPT               00000041               DAP$K_BLN                  000000C0
DAP$B_AID                  00000050               DAP$K_CMP_MSG            = 00000007
DAP$B_ALN                  00000044               DAP$K_CRC_INIT           = 0000FFFF
DAP$B_AOP                  00000045               DAP$K_CREATE             = 00000002
DAP$B_BITCNT               00000035               DAP$K_DAT_MSG            = 00000008
DAP$B_BKS                  00000050               DAP$K_FIX                = 00000001
DAP$B_BKZ                  00000051               DAP$K_RESPONSE           = 00000002
DAP$B_BLKCNT               00000056               DAP$K_SEQ                = 00000000
DAP$B_BSZ                  00000052               DAP$K_SEQ_ACC            = 00000000
DAP$B_CMPFUNC              00000040               DAP$K_STG                = 00000000
DAP$B_CONFUNC              00000040               DAP$K_SUBMIT             = 00000007
DAP$B_CTLFUNC              00000040               DAP$L_ALQ1                 0000004C
DAP$B_DAN                  00000070               DAP$L_ALQ2                 0000004C
DAP$B_DATATYPE             00000044               DAP$L_ATTMENU              00000040
DAP$B_DBS                  0000007C               DAP$L_CMWA                 00000030
DAP$B_DCODE_FID            00000019               DAP$L_CRC_RSLT             00000020
DAP$B_DCODE_MAC            0000001B               DAP$L_DCODE_STS            00000018
DAP$B_DCODE_MSG            0000001A               DAP$L_DEV                  00000068
DAP$B_DECVER               00000047               DAP$L_DVB                  00000078
DAP$B_DTP                  00000071               DAP$L_EBK                  00000078
DAP$B_ECONUM               00000045               DAP$L_FOP1                 00000064
DAP$B_FAC                  00000042               DAP$L_FOP2                 00000044
DAP$B_FILESYS              00000043               DAP$L_HBK                  00000074
DAP$B_FLAGS                00000031               DAP$L_KEYMENU              00000040
DAP$B_FLG                  00000048               DAP$L_LOC                  00000048
DAP$B_FSZ                  00000051               DAP$L_MRN                  00000058
DAP$B_IAN                  0000006E               DAP$L_MSG_MASK             0000001C
DAP$B_IBS                  0000007D               DAP$L_RECNUM1              00000040
DAP$B_KRF                  00000047               DAP$L_RECNUM2              00000048
DAP$B_LAN                  0000006F               DAP$L_ROP                  00000050
DAP$B_LEN256               00000034               DAP$L_RVB                  00000074
DAP$B_LENGTH               00000033               DAP$L_SBN                  0000007C
DAP$B_LVL                  0000007E               DAP$L_SSPWA                00000080
DAP$B_NAMETYPE             00000040               DAP$L_STV                  0000004C
DAP$B_NOA                  00000045               DAP$L_TEMP                 00000090
DAP$B_NOK                  00000044               DAP$M_BITCNT             = 00000008
DAP$B_NOR                  00000046               DAP$M_BLKCNT             = 00000040
DAP$B_NSG                  00000049               DAP$M_CMPFMT             = 00000008
```

```
DAP$M_DFTSPEC            = 00000010         DAP$V_X_CHECK          = 00000001
DAP$M_DIRNAME            = 00000004         DAP$V_X_RECNUM         = 00000000
DAP$M_DMO                = 00002000         DAP$W_ALLMENU            00000040
DAP$M_DSP_3NAM           = 00000200         DAP$W_BLS                00000048
DAP$M_DSP_ALL            = 00000004         DAP$W_BUFSIZ             00000040
DAP$M_DSP_KEY            = 00000002         DAP$W_CHECK              00000042
DAP$M_DSP_NAM            = 00000100         DAP$W_CTLMENU            00000044
DAP$M_EMBEDDED           = 00000010         DAP$W_DEQ1               00000054
DAP$M_FILNAME            = 00000002         DAP$W_DEQ2               00000052
DAP$M_GET                = 00000002         DAP$W_DFL                00000044
DAP$M_GO_NOGO            = 00000010         DAP$W_DISPLAY1           0000004C
DAP$M_IMAGE              = 00000002         DAP$W_DISPLAY2           00000054
DAP$M_LSA                = 00000040         DAP$W_FFB                00000072
DAP$M_MACY11             = 00000080         DAP$W_IFL                00000046
DAP$M_MSE                = 00000010         DAP$W_LRL                00000070
DAP$M_SEGMENT            = 00000040         DAP$W_MRL                00000072
DAP$M_TMP1$              = 00000020         DAP$W_MRS                0000004A
DAP$M_TMP2$              = 000000C0         DAP$W_PARTNER            00000006
DAP$M_TMP3$              = 00020000         DAP$W_POS                0000004C
DAP$M_TMP4$              = 01000000         DAP$W_POS_TMP            0000004A
DAP$M_TMP5$              = F0000000         DAP$W_PROGRP             00000054
DAP$M_VOLNAME            = 00000008         DAP$W_PROMENU            00000040
DAP$M_ZERO               = 00000080         DAP$W_PROOWN             00000052
DAP$Q_ADT                  00000070         DAP$W_PROSYS             00000050
DAP$Q_BDT                  00000060         DAP$W_PROWLD             00000056
DAP$Q_CDT                  00000048         DAP$W_PVN                00000042
DAP$Q_DCODE_FLG            00000000         DAP$W_RFA                00000042
DAP$Q_EDT                  00000058         DAP$W_RVN                00000042
DAP$Q_FILEDATA             00000044         DAP$W_STSCODE            00000040
DAP$Q_FILESPEC             00000044         DAP$W_SUMENU             00000040
DAP$Q_KEY                  00000048         DAP$W_TIMENU             00000040
DAP$Q_KNM                  00000064         DAP$W_VERSION            00000004
DAP$Q_MSG_BUF1             00000008         DAP$W_VOL                00000042
DAP$Q_MSG_BUF2             00000010         DAP$_ACCFUNC           = 00000010
DAP$Q_NAMESPEC             00000044         DAP$_CMPFUNC           = 00000010
DAP$Q_OWNER                00000048         DAP$_CONFUNC           = 00000010
DAP$Q_PASSWORD             00000050         DAP$_CTLFUNC           = 00000010
DAP$Q_PDT                  00000068         DAP$_FILE_CLOS         = 00000007
DAP$Q_RDT                  00000050         DAP$_FILE_OPEN         = 00000004
DAP$Q_RUNSYS               0000005C         DAP$_FILE_XFER         = 00000005
DAP$Q_STX                  00000050         DAP$_INVALID           = 00000009
DAP$Q_SYSCAP               00000028         DAP$_KEY               = 00000013
DAP$Q_SYSPEC               00000038         DAP$_MSG_SYNC          = 0000000A
DAP$V_CHGTIMCLS          = 0000002B         DAP$_SUCCESS           = 00000001
DAP$V_DSP_ALL            = 00000002         DAP$_UNSUPPORT         = 00000002
DAP$V_DSP_ATT            = 00000000         DC1                    = 00000011
DAP$V_DSP_KEY            = 00000001         DC2                    = 00000012
DAP$V_DSP_NAM            = 00000008         DC3                    = 00000013
DAP$V_DSP_PRO            = 00000005         DC4                    = 00000014
DAP$V_DSP_SUM            = 00000003         DLE                    = 00000010
DAP$V_DSP_TIM            = 00000004         ERR_FILE                 00000CA7 R     03
DAP$V_GEQ_V42            = 00000021         ERR_FILE_CLOS            00000CA0 R     03
DAP$V_GEQ_V56            = 00000024         ERR_FILE_OPEN            00000C8E R     03
DAP$V_GEQ_V70            = 00000026         ERR_FILE_XFER            00000C97 R     03
DAP$V_RET_CRC            = 00000003         ERR_INVALID              00000CB3 R     03
DAP$V_VAXVMS             = 00000034         ERR_SYNCHRONIZE          00000CC5 R     03
DAP$V_WILDCARD           = 00000026         ERR_UNSUPPORT            00000CBC R     03
```

```
ESC                            = 0000001B                      FAL$CHECK_NMF                    000000AC RG      03
EXIT_FAILURE                     00000CD6 R        03          FAL$CHECK_WILD                   00000078 RG      03
EXIT_STS_ALWAYS                  0000096C R        03          FAL$CLOSE                        000009AC RG      03
EXIT_STS_FAIL                    00000CCC R        03          FAL$CONNECT                      0000041E RG      03
EXIT_STS_SUC                     00000CD9 R        03          FAL$CRC_LOGERR                   ********    X    03
EXIT_SUCCESS                     00000CE6 R        03          FAL$CRC_TABLE                    ********    X    03
FAB$B_DNS                      = 00000035                      FAL$CREATE                       0000011B RG      03
FAB$B_FNS                      = 00000034                      FAL$C_STBBLN                     00000040
FAB$B_FSZ                      = 0000003F                      FAL$C_WRKBLN                     00002000
FAB$B_ORG                      = 0000001D                      FAL$DELETE                       00000890 RG      03
FAB$B_RFM                      = 0000001F                      FAL$DIR_END                      000002F8 RG      03
FAB$B_SHR                      = 00000017                      FAL$DIR_PARSE                    000002C2 RG      03
FAB$C_FIX                      = 00000001                      FAL$DIR_SEARCH                   000002FE RG      03
FAB$C_IDX                      = 00000020                      FAL$DISCARD_DAT                  00000846 RG      03
FAB$C_SEQ                      = 00000000                      FAL$DISCONNECT                   00000975 RG      03
FAB$C_STM                      = 00000004                      FAL$DISPLAY                      000008CA RG      03
FAB$C_STMCR                    = 00000006                      FAL$ENCODE_ALL                   ********    X    03
FAB$C_STMLF                    = 00000005                      FAL$ENCODE_ATT                   ********    X    03
FAB$C_UDF                      = 00000000                      FAL$ENCODE_KEY                   ********    X    03
FAB$C_VAR                      = 00000002                      FAL$ENCODE_NAM                   ********    X    03
FAB$C_VFC                      = 00000003                      FAL$ENCODE_NAM1                  ********    X    03
FAB$L_DNA                      = 00000030                      FAL$ENCODE_PRO                   ********    X    03
FAB$L_FNA                      = 0000002C                      FAL$ENCODE_STS                   ********    X    03
FAB$L_FOP                      = 00000004                      FAL$ENCODE_SUM                   ********    X    03
FAB$L_NAM                      = 00000028                      FAL$ENCODE_TIM                   ********    X    03
FAB$L_STS                      = 00000008                      FAL$ERASE                        0000017C RG      03
FAB$L_STV                      = 0000000C                      FAL$EXECUTE                      00000268 RG      03
FAB$L_XAB                      = 00000024                      FAL$EXTEND                       000008F7 RG      03
FAB$M_DLT                      = 00008000                      FAL$FIL_PARSE                    0000004D RG      03
FAB$M_SCF                      = 00004000                      FAL$FIL_SEARCH                   0000007E RG      03
FAB$M_SHRGET                   = 00000002                      FAL$FIND                         000008A9 RG      03
FAB$M_SHRPUT                   = 00000001                      FAL$FLUSH                        00000934 RG      03
FAB$M_SPL                      = 00002000                      FAL$FREE                         0000093F RG      03
FAB$M_UPI                      = 00000040                      FAL$GET_FTM                      0000051B RG      03
FAB$V_DLT                      = 0000000F                      FAL$GET_RAM                      000004A0 RG      03
FAB$V_OFP                      = 0000001D                      FAL$GL_ORITWAIT                  ********    X    03
FAB$V_SCF                      = 0000000E                      FAL$GQ_WILDSPEC                  ********    X    03
FAB$W_BLS                      = 0000003C                      FAL$INIT_ALLXAB                  ********    X    03
FAB$W_DEQ                      = 00000014                      FAL$INIT_DATXAB                  ********    X    03
FAB$W_IFI                      = 00000002                      FAL$INIT_KEYXAB                  ********    X    03
FAB$W_MRS                      = 00000036                      FAL$INIT_PROXAB                  ********    X    03
FAL$BIT_BUCKET                   00000840 RG       03          FAL$INIT_SUMXAB                  ********    X    03
FAL$BUILD_HEAD                   ********    X     03          FAL$INTE_MSG                     00000025 RG      03
FAL$BUILD_TAIL                   ********    X     03          FAL$INV_ACCFUNC                  00000A16 RG      03
FAL$B_ACCFUNC                    000001F6                      FAL$INV_CMPFUNC                  00000A34 RG      03
FAL$B_ACCOPT                     000001F5                      FAL$INV_CONFUNC                  00000A22 RG      03
FAL$B_DATATYPE                   000001F4                      FAL$INV_CTLFUNC                  00000A1C RG      03
FAL$B_DISABLE                    00000006                      FAL$K_ALLXAB                   = 00000020
FAL$B_ENABLE                     00000005                      FAL$K_FAB                      = 00000050
FAL$B_LOGGING                    00000004                      FAL$K_KEYXAB                   = 0000004C
FAL$B_MISCOPT                    00000007                      FAL$K_MAX_AID                  = 0000001F
FAL$B_RAC                        000001F7                      FAL$K_MAX_REF                  = 0000001F
FAL$B_RBK_CACHE                  00000012                      FAL$K_STBBLN                     00000040
FAL$B_RCVBUFIDX                  00000011                      FAL$K_WRKBLN                     00002000
FAL$B_VALUE                      00000010                      FAL$LOAD_IMAGE                   000003CB RG      03
FAL$CHANGE                       000009A3 RG       03          FAL$LOG_CLSMSG                   ********    X    03
FAL$CHECK_FTM                    0000043A RG       03          FAL$LOG_RESNAM                   ********    X    03
```

FAL
V04

```
FAL$L_ALLXAB                    00000C00
FAL$L_ALLXABINI                 00000074
FAL$L_CHAIN_NXT                 0000007C
FAL$L_DATXAB                    00000320
FAL$L_FAB                       00000200
FAL$L_FAB2                      00000800
FAL$L_FHCXAB                    000002F4
FAL$L_FOP                       000001F8
FAL$L_KEYNAM                    00001C00
FAL$L_KEYXAB                    00001000
FAL$L_KEYXABINI                 00000078
FAL$L_NAM                       00000294
FAL$L_NAM2                      00000850
FAL$L_NUMBER                    000001FC
FAL$L_PROXAB                    0000034C
FAL$L_RAB                       00000250
FAL$L_RCVBUF                    0000005C
FAL$L_RCV_DAT                   00000008
FAL$L_RCV_LNK                   00000010
FAL$L_RCV_MSG                   00000004
FAL$L_RCV_PKT                   00000000
FAL$L_RCV_USR                   0000000C
FAL$L_RDTXAB                    000003B0
FAL$L_RMS_PTR                   0000006C
FAL$L_STB                       000000C0
FAL$L_SUMXAB                    000003A4
FAL$L_TEMP                      000003F4
FAL$L_USE_SC1                   000000A8
FAL$L_USE_SC2                   000000AC
FAL$L_USE_VER                   000000A4
FAL$L_XMT_DAT                   0000001C
FAL$L_XMT_LNK                   00000024
FAL$L_XMT_MSG                   00000018
FAL$L_XMT_PKT                   00000014
FAL$L_XMT_USR                   00000020
FAL$MBX_RCV_QIO                 ********     X  03
FAL$M_DATXAB                  = 00000004
FAL$NEXT_MSG                    00000000 RG     03
FAL$OPEN                        000000BA RG     03
FAL$OUT_OF_SEQ                  00000A13 RG     03
FAL$PUT_FTM                     000006FE RG     03
FAL$PUT_RAM                     000006A4 RG     03
FAL$Q_BCD                       00000050
FAL$Q_DIRNAME                   00000088
FAL$Q_FALLOG                    00000090
FAL$Q_FLG                       00000000
FAL$Q_MBX                       00000038
FAL$Q_MBXIOSB                   00000030
FAL$Q_RCV                       00000040
FAL$Q_RCVIOSB                   00000020
FAL$Q_RMS                       00000064
FAL$Q_STATE_CTX                 00000008
FAL$Q_SYSNET                    00000098
FAL$Q_TEMP                      000003F8
FAL$Q_VOLNAME                   00000080
FAL$Q_XMT                       00000048
FAL$Q_XMTIOSB                   00000028
FAL$READ_FTM                    000005C8 RG     03
FAL$READ_RAM                    0000057B RG     03
FAL$RECEIVE                     ********     X  03
FAL$RECEIVE_MBX                 ********     X  03
FAL$RELEASE                     0000094A RG     03
FAL$RENAME                      000001B1 RG     03
FAL$RESET                       0000098E RG     03
FAL$RETRV_FTM                   0000044A RG     03
FAL$RETRV_RAM                   00000440 RG     03
FAL$REWIND                      0000091E RG     03
FAL$SAVE_MSG                    00000040 RG     03
FAL$SPACE_BW                    00000955 RG     03
FAL$SPACE_FW                    0000095A RG     03
FAL$STM_MASK                    00000000 RG     01
FAL$STORE_END                   000007FF RG     03
FAL$STORE_FTM                   0000067B RG     03
FAL$STORE_RAM                   00000671 RG     03
FAL$SUBMIT                      00000113 RG     03
FAL$TEST_MSG                    0000003A RG     03
FAL$TRANSMIT                    ********     X  03
FAL$TRUNCATE                    00000929 RG     03
FAL$T_DAP                       00000100
FAL$T_DIRNAME                   00001F00
FAL$T_EXPAND                    00000500
FAL$T_EXPAND2                   00000A00
FAL$T_FALLOG                    00001C00
FAL$T_FILESPEC                  00000400
FAL$T_FILESPEC2                 00000900
FAL$T_KEYBUF                    00000700
FAL$T_MBXBUF                    00001980
FAL$T_PRTBUF1                   00001A00
FAL$T_PRTBUF2                   00001B00
FAL$T_RESULT                    00000600
FAL$T_RESULT2                   00000B00
FAL$T_SYSNET                    00001D00
FAL$T_VOLNAME                   00001E00
FAL$UNS_ACCFUNC                 00000A28 RG     03
FAL$UNS_CONFUNC                 00000A2E RG     03
FAL$UNS_KEY                     00000A3A RG     03
FAL$UPDATE                      00000850 RG     03
FAL$V_ALLXAB                  = 00000001
FAL$V_ATT_MSG                 = 00000001
FAL$V_BLK_IO                  = 00000009
FAL$V_DATXAB                  = 00000002
FAL$V_DIS_PMR                 = 00000033
FAL$V_DIS_RBK                 = 00000032
FAL$V_FTM                     = 00000008
FAL$V_KEYXAB                  = 00000000
FAL$V_LAST_MSG                = 00000018
FAL$V_MBXAST                  = 00000015
FAL$V_NEWNAM                  = 0000000B
FAL$V_PROXAB                  = 00000003
FAL$V_RCVAST                  = 00000011
FAL$V_RDTXAB                  = 00000004
FAL$V_RET_RECN                = 0000001A
FAL$V_RET_RFA                 = 00000019
FAL$V_RET_STV                 = 0000001B
```

```
FAL$V_WILD          = 0000000A               RMS$_WLD            = 00018744
FAL$WRITE_FTM         00000768 RG    03      SEND_3PART_NAM        00000C16 R    03
FAL$WRITE_RAM         00000733 RG    03      SEND_ACK              00000B62 R    03
FAL$W_DAPBUFSIZ       0000001A               SEND_ATTRIBUTES       00000339 R    03
FAL$W_DISPLAY         00000070               SEND_CMP              00000B6F R    03
FAL$W_LNKCHN          0000001C               SEND_DIR_NAM          00000C4B R    03
FAL$W_MBXCHN          0000001E               SEND_FIL_NAM          00000C72 R    03
FAL$W_QIOBUFSIZ       00000018               SEND_OPTNL_LOOP       00000B9B R    03
FAL$W_RECEIVED        00000072               SEND_OPTNL_MSGS       00000B83 R    03
FAL$W_USE_DBS         000000A0               SEND_VOL_NAM          00000C1C R    03
FAL$W_USE_SYS         000000A2               SS$_ACCONFLICT      = 00000800
FF                  = 0000000C               SS$_FCPREADERR      = 00000888
LF                  = 0000000A               SS$_FILELOCKED      = 000008A8
MAP_SS_TO_RMS         00000399 R     03      SS$_NOPRIV          = 00000024
NAM$B_DEV           = 00000039               SYS$CLOSE             ********  GX  03
NAM$B_DIR           = 0000003A               SYS$CONNECT           ********  GX  03
NAM$B_ESL           = 0000000B               SYS$CREATE            ********  GX  03
NAM$B_NAME          = 0000003B               SYS$DELETE            ********  GX  03
NAM$B_NODE          = 00000038               SYS$DISCONNECT        ********  GX  03
NAM$B_NOP           = 00000008               SYS$DISPLAY           ********  GX  03
NAM$B_RSL           = 00000003               SYS$ERASE             ********  GX  03
NAM$B_TYPE          = 0000003C               SYS$EXTEND            ********  GX  03
NAM$B_VER           = 0000003D               SYS$FIND              ********  GX  03
NAM$L_DIR           = 00000048               SYS$FLUSH             ********  GX  03
NAM$L_ESA           = 0000000C               SYS$FREE              ********  GX  03
NAM$L_FNB           = 00000034               SYS$GET               ********  GX  03
NAM$L_NAME          = 0000004C               SYS$OPEN              ********  GX  03
NAM$L_NODE          = 00000040               SYS$PARSE             ********  GX  03
NAM$L_RLF           = 00000010               SYS$PUT               ********  GX  03
NAM$L_RSA           = 00000004               SYS$READ              ********  GX  03
NAM$V_NODE          = 00000011               SYS$RELEASE           ********  GX  03
NAM$V_SRCHXABS      = 00000006               SYS$RENAME            ********  GX  03
NAM$V_WILDCARD      = 00000008               SYS$REWIND            ********  GX  03
RAB$B_RAC           = 0000001E               SYS$SEARCH            ********  GX  03
RAB$C_KEY           = 00000001               SYS$SPACE             ********  GX  03
RAB$L_BKT           = 00000038               SYS$TRUNCATE          ********  GX  03
RAB$L_RBF           = 00000028               SYS$UPDATE            ********  GX  03
RAB$L_RHB           = 0000002C               SYS$WAIT              ********  GX  03
RAB$L_ROP           = 00000004               SYS$WRITE             ********  GX  03
RAB$L_STS           = 00000008               VT                  = 0000000B
RAB$L_STV           = 0000000C               WRITE_FTM_NORBK       000007E2 R    03
RAB$L_UBF           = 00000024               WRITE_FTM_RBK         0000077E R    03
RAB$V_ASY           = 00000000               XAB$B_COD           = 00000000
RAB$W_RSZ           = 00000022               XAB$B_NOA           = 00000008
RAB$W_USZ           = 00000020               XAB$B_NOK           = 00000009
READ_FTM_NORBK        00000638 R     03      XAB$C_RDT           = 0000001E
READ_FTM_RBK          000005D7 R     03      XAB$L_NXT           = 00000004
RMS$_ACC            = 0001C002
RMS$_ATR            = 0001C0CC
RMS$_CRC            = 000182E2
RMS$_EOF            = 0001827A
RMS$_FLK            = 0001828A
RMS$_FNF            = 00018292
RMS$_NMF            = 000182CA
RMS$_NOD            = 000185F4
RMS$_PRV            = 0001829A
RMS$_RFM            = 00018664
```

```
                                    +----------------+
                                    ! Psect synopsis !
                                    +----------------+

PSECT name                Allocation          PSECT No.  Attributes
----------                ----------          ---------  ----------
.  ABS  .                 00000000  (    0.)  00 (  0.)  NOPIC   USR   CON   ABS   LCL NOSHR NOEXE NORD   NOWRT NOVEC BYTE
FAL$DATA                  00000004  (    4.)  01 (  1.)  NOPIC   USR   CON   REL   LCL   SHR NOEXE   RD     WRT NOVEC LONG
$ABS$                     00002000  ( 8192.)  02 (  2.)  NOPIC   USR   CON   ABS   LCL NOSHR   EXE   RD     WRT NOVEC BYTE
FAL$CODE                  00000CEA  ( 3306.)  03 (  3.)  NOPIC   USR   CON   REL   LCL NOSHR   EXE   RD   NOWRT NOVEC BYTE

                                  +------------------------+
                                  ! Performance indicators !
                                  +------------------------+

Phase                    Page faults    CPU Time       Elapsed Time
-----                    -----------    --------       ------------
Initialization                   35    00:00:00.06    00:00:00.69
Command processing              118    00:00:00.39    00:00:02.38
Pass 1                          731    00:00:23.51    00:01:33.79
Symbol table sort                 0    00:00:02.76    00:00:11.65
Pass 2                          412    00:00:05.84    00:00:24.98
Symbol table output               1    00:00:00.29    00:00:00.41
Psect synopsis output             0    00:00:00.01    00:00:00.01
Cross-reference output            0    00:00:00.00    00:00:00.00
Assembler run totals           1299    00:00:32.87    00:02:13.92

The working set limit was 2550 pages.
194390 bytes (380 pages) of virtual memory were used to buffer the intermediate code.
There were 140 pages of symbol table space allocated to hold 2683 non-local and 183 local symbols.
2604 source lines were read in Pass 1, producing 29 object records in Pass 2.
75 pages of virtual memory were used to define 72 macros.

                                  +--------------------------+
                                  ! Macro library statistics !
                                  +--------------------------+

Macro library name                        Macros defined
------------------                        --------------
_$255$DUA28:[FAL.OBJ]FAL.MLB;1                  25
_$255$DUA28:[SYSLIB]STARLET.MLB;2               44
TOTALS (all libraries)                          69

3282 GETS were required to define 69 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LIS$:FALACTION/OBJ=OBJ$:FALACTION MSRC$:FALACTION/UPDATE=(ENH$:FALACTION)+LIB$:FAL/LIB
```

0174 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

FALACTION
LIS

FALDAPCRC
LIS

FALMACROS
MAR

FALBLDXAB
LIS

FALBLDATT
LIS

FALBLDSTS
LIS

FALDEF
MDL

FALDAPIO
LIS

FALACTINI
LIS

FALACTMSG
LIS