


```

RRRRRRRR      MM      MM      SSSSSSSS      000000      EEEEEEEEEEE      XX      XX      TTTTTTTTTT      EEEEEEEEEEE      NN      NN
RRRRRRRR      MM      MM      SSSSSSSS      000000      EEEEEEEEEEE      XX      XX      TTTTTTTTTT      EEEEEEEEEEE      NN      NN
RR      RR      MMMM      MMMM      SS      00      00      EE      XX      XX      TT      EE      NN      NN
RR      RR      MMMM      MMMM      SS      00      00      EE      XX      XX      TT      EE      NN      NN
RR      RR      MM      MM      SS      00      0000      EE      XX      XX      TT      EE      NNNN      NN
RR      RR      MM      MM      SS      00      0000      EE      XX      XX      TT      EE      NNNN      NN
RRRRRRRR      MM      MM      SSSSSS      00      00      EEEEEEEEE      XX      NN      NN
RRRRRRRR      MM      MM      SSSSSS      00      00      EEEEEEEEE      XX      NN      NN
RR      RR      MM      MM      SS      0000      00      EE      XX      NN      NN
RR      RR      MM      MM      SS      0000      00      EE      XX      NN      NN
RR      RR      MM      MM      SS      00      00      EE      XX      NN      NN
RR      RR      MM      MM      SS      00      00      EE      XX      NN      NN
RR      RR      MM      MM      SSSSSSSS      000000      EEEEEEEEEEE      XX      XX      TT      EEEEEEEEEEE      NN      NN
RR      RR      MM      MM      SSSSSSSS      000000      EEEEEEEEEEE      XX      XX      TT      EEEEEEEEEEE      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
LLLLLLLLLLLL      IIIIII      SSSSSSSS
LLLLLLLLLLLL      IIIIII      SSSSSSSS

```

(3) 85
(4) 123
(7) 296
(8) 376
(13) 538

DECLARATIONS
RMS\$EXTEND - \$EXTEND ROUTINE
RMS\$ALLOC_BUF - CO-ROUTINE TO ALLOCATE/DEALLOCATE BUFFERS
RMS\$EXTEND_XAB - ROUTINE TO HANDLE ALLOCATION XABS
RMS\$CARVE_BDB


```
0000 1          $BEGIN RMSOEXTEN,000,RM$RMS,<DISPATCH FOR EXTEND OPERATION>
0000 2
0000 3
0000 4 :*****
0000 5 :*
0000 6 :*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0000 7 :*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0000 8 :*  ALL RIGHTS RESERVED.
0000 9 :*
0000 10 :*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0000 11 :*  ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0000 12 :*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0000 13 :*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0000 14 :*  OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0000 15 :*  TRANSFERRED.
0000 16 :*
0000 17 :*  THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0000 18 :*  AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0000 19 :*  CORPORATION.
0000 20 :*
0000 21 :*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0000 22 :*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0000 23 :*
0000 24 :*
0000 25 :*****
```



```
0000 27 :++
0000 28 :
0000 29 : Facility: rms32
0000 30 :
0000 31 : Abstract:
0000 32 :         this module is the highest level control routine
0000 33 :         to perform the $extend function.
0000 34 :
0000 35 : Environment:
0000 36 :         star processor running starlet exec.
0000 37 :
0000 38 : Author:      L F Laverdure,   creation date: 11-JAN-1978
0000 39 :
0000 40 : Modified By:
0000 41 :
0000 42 :         V03-012 RAS0284      Ron Schaefer      29-Mar-1984
0000 43 :         Fix error paths to put the area id in the STV.
0000 44 :
0000 45 :         V03-011 SHZ0001      Stephen H. Zalewski, 13-Mar-1984
0000 46 :         When you allocate a BDB, you must bump the AVLCL counter.
0000 47 :
0000 48 :         V03-010 DAS0001      David Solomon      12-Nov-1983
0000 49 :         Fix incorrect register use (R5 instead of R6) in RM$CARVE_BDB.
0000 50 :
0000 51 :         V03-009 CWH3009      CW Hobbs          29-Oct-1983
0000 52 :         Fix two uses of R9 to R10 to correct an accvio on
0000 53 :         an ISAM file path.
0000 54 :
0000 55 :         V03-008 KPL0001      Peter Lieberwirth  27-Oct-1983
0000 56 :         To journal extends before a connect is done the
0000 57 :         BDB and buffer allocated need to contain extra BDBs
0000 58 :         and buffers.
0000 59 :
0000 60 :         V03-007 KBT0543      Keith B. Thompson  10-Jun-1983
0000 61 :         Fix broken branch
0000 62 :
0000 63 :         V03-006 RAS0140      Ron Schaefer      24-Mar-1983
0000 64 :         Fix bugcheck caused by forgotten BLB for relative
0000 65 :         and shared sequential files if an error occurs on
0000 66 :         the actual extend (diskquota probably).
0000 67 :
0000 68 :         V03-004 KBT0315      Keith B. Thompson  8-Sep-1982
0000 69 :         Remove all of the old S0 sharing code
0000 70 :
0000 71 :         V03-003 KBT0181      Keith B. Thompson  23-Aug-1982
0000 72 :         Reorganize psects and rename entry point to single '$'
0000 73 :
0000 74 :         V03-002 KBT0091      Keith B. Thompson  13-Jul-1982
0000 75 :         Clean up psects
0000 76 :
0000 77 :         V03-001 CDS0003      C Saether        30-Mar-1982
0000 78 :         Always allocate buffer if block i/o in
0000 79 :         RM$ALLOC_BUF routine.
0000 80 :
0000 81 : --
0000 82 :
0000 83 :
```

```
0000 85      .SBTTL  DECLARATIONS
0000 86
0000 87      :
0000 88      : Include Files:
0000 89      :
0000 90      :
0000 91      :
0000 92      : Macros:
0000 93      :
0000 94      :
0000 95      $CSHDEF
0000 96      $DEVDEF
0000 97      $FABDEF
0000 98      $IFBDEF
0000 99      $IRBDEF
0000 100     $RLSDEF
0000 101     $XABDEF
0000 102     $XABALLDEF
0000 103     $BDBDEF
0000 104     $RJRDEF
0000 105     $RMSDEF
0000 106
0000 107     :
0000 108     : Equated Symbols:
0000 109     :
0000 110
00000020 0000 111     FOP=FAB$L_FOP*8
0000 112
0000 113     :
0000 114     : Own Storage:
0000 115     :
0000 116     :
0000 117     : argument list for xab chain processing (allocation xabs only)
0000 118     :
0000 119     :
00 00'20 14 0000 120 XAB_ARGS:
0000 121     .BYTE  XAB$C_ALL,XAB$C_ALLLEN,XBC$C_EXTALL,0
```



```
0004 123      .SBTTL RMS$EXTEND - $EXTEND ROUTINE
0004 124
0004 125      :++
0004 126      :
0004 127      RMS$EXTEND
0004 128
0004 129      this routine performs the highest level $extend processing.
0004 130      its functions include:
0004 131
0004 132      1. common setup
0004 133      2. check for all streams idle, exiting if not
0004 134      3. perform validity checking
0004 135      4. call main body of extend logic (extend_it) subroutine - first thing
0004 136      it must do is to call the co-routine alloc_buf to allocate
0004 137      buffer and lock bdb if necessary. it does co-routine call back
0004 138      if buffers allocated so that it can return them when extend_it
0004 139      exits, else does rsb (will not return to it) if no action required.
0004 140      5. lock prolog for rel. f. o. not block i/o accessed
0004 141      6. process any allocation xabs, performing the indicated extends
0004 142      and bucket formatting if relative.
0004 143      isam subroutine in rm3face will be called to do the isam stuff.
0004 144      7. if no allocation xabs present, perform the extend based upon the fab.
0004 145      8. release the prolog if locked (relative f.o.)
0004 146      9. exit from extend it may be call back to alloc_buf to cleanup
0004 147      buffers if allocated - this preserves error code in r0.
0004 148      10. exit to the user, generating an ast if requested
0004 149
0004 150
0004 151      Calling sequence:
0004 152
0004 153      entered from exec as a result of user's calling sys$extend
0004 154      (e.g., by using the $extend macro).
0004 155
0004 156      Input Parameters:
0004 157
0004 158      ap      user's argument list addr
0004 159
0004 160      Implicit Inputs:
0004 161
0004 162      the contents of the fab and possible related user interface
0004 163      blocks.
0004 164
0004 165      Output Parameters:
0004 166
0004 167      r0      status code
0004 168      r1      destroyed
0004 169
0004 170      Implicit Outputs:
0004 171
0004 172      the size of the extension is returned in fab$l_alq or xab$l_alq
0004 173      if allocation xab(s) present.
0004 174
0004 175      a completion ast is queued if so specified by the user.
0004 176
0004 177      Completion Codes:
0004 178
0004 179      standard rms (see functional spec for list).
```

```
0004 180 :  
0004 181 : Side Effects:  
0004 182 :  
0004 183 : any locate mode pointer for the file is invalidated.  
0004 184 :  
0004 185 :--  
0004 186 :  
0004 187 :++  
0004 188 :  
0004 189 : extended branches for error conditions  
0004 190 :  
0004 191 :--  
0004 192 :  
FFF9' 31 0004 193 ERRIOP: BRW RMSERRIOP ; extend on non-disk device  
FFF6' 31 0007 194 ERRFAC: BRW RM$ERRFAC ; file not write accessed  
000A 195 :  
000A 196 :++  
000A 197 :  
000A 198 : entry point for $extend service  
000A 199 :  
000A 200 :--  
000A 201 :  
000A 202 $ENTRY RMS$EXTEND  
000A 203 $TSTPT EXTEND  
FFED' 30 0010 204 BSBW RM$FSET ; do common setup  
0013 205 : note: does not return on error  
06 6A 3E E1 0013 206 BBC #IFB$V_DAP,(R10),10$ ; Check for network operation  
FFE6' 30 0017 207 BSBW NT$EXTEND ; Extend file on remote system  
FFE3' 31 001A 208 BRW RM$EXRMS ; and exit RMS  
001D 209 :  
001D 210 :  
001D 211 : Check that device is disk and that we are write accessed  
001D 212 :  
001D 213 :  
E3 6A 1C E1 001D 214 10$: BBC #DEV$V_RND,IFB$L PRIM_DEV(R10),ERRIOP; branch if not disk  
E2 6A 30 E1 0021 215 BBC #IFB$V_WRTACC,(R10),ERRFAC; or if not write accessed  
03 10 0025 216 BSBB EXTEND-IT ; call extend routine  
FFD6' 31 0027 217 BRW RM$EXRMS ; and exit rms
```



```
002A 219
002A 220 :
002A 221 : main body of extend code to perform extend function.
002A 222 : first call alloc_buf which will allocate buffer and necessary control
002A 223 : blocks if no streams are connected. if a stream is already connected,
002A 224 : no action is necessary as buffer and control blocks will already be present.
002A 225 :
002A 226
002A 227 EXTEND_IT:
31 22 AA 05 E0 002A 228 BBS #IFB$V_BIO,IFB$B_FAC(R10),GETXAB; if block i/o, just go
002F 229 ; direct to allocate space to
002F 230 ; file. no checks on rel or idx
002F 231
002F 232 ASSUME IFB$C_SEQ EQ 0
002F 233
23 AA 95 002F 234 TSTB IFB$B_ORGCASE(R10) ; sequential file ?
2C 13 0032 235 BEQL GETXAB ; if seq, skip buffer alloc
55 5E AA 9A 0034 236 MOVZBL IFB$B_BKS(R10), R5 ; bucket size in blocks for alloc_buf
56 01 D0 0038 237 MOVL #1, R6 ; needs a lock blb.
52 10 003B 238 BSBB RM$ALLOC_BUF ; see comments above
4E 50 E9 003D 239 BLBC R0, RETURN ; out on error
23 AA 01 91 0040 240 CMPB #IFB$C_REL, IFB$B_ORGCASE(R10); relative file ?
08 12 0044 241 BNEQ 10$ ; branch if isam file
00000000'EF 16 0046 242 JSB RM$LOCK_PROLOG ; read and lock prolog for rel
0B 11 004C 243 BRB 20$ ; join rest of code
52 D4 004E 244 10$: CLRL R2 ; zero buffer size
0050 245 $CACHE VBN=#1,- ; lock vbn 1 to extend
0050 246 FLAGS=<LOCK,NOREAD,NOBUFFER>
6C AA 32 50 E9 0059 247 20$: BLBC R0, RETURN ; exit on error
54 D0 005C 248 MOVL R4, IFB$L_LOCK_BDB(R10) ; save bdb address
0060 249
0060 250 :
0060 251 : process allocation xab(s), if any.
0060 252 :
0060 253 : the subroutine rm$extend_xab is called for each allocation xab found.
0060 254 :
0060 255 :
0060 256
5C 9D AF 9E 0060 257 GETXAB: MOVAB XAB_ARGS, AP ; set arg list addr
FF99' 30 0064 258 BSBW RM$XAB_SCAN ; go look for xab
0C 50 E9 0067 259 BLBC R0, EXIT ; branch on error
54 D5 006A 260 TSTL R4 ; any xabs found?
08 12 006C 261 BNEQ EXIT ; branch if yes, allocation
006E 262 ; occurred processing xab(s)
006E 263 :
006E 264 : there were no allocation xabs found.
006E 265 : use the alq field of the fab to specify the size of the allocation.
006E 266 :
006E 267
53 58 D0 006E 268 MOVL R8, R3 ; cause fab alq to be used
0071 269 ; xab, fab inputs at same offset
56 D4 0071 270 CLRL R6 ; say no xab
00B1 30 0073 271 BSBW XTND ; go do the extend
0076 272
0076 273 ASSUME IFB$C_SEQ EQ 0
0076 274
23 AA 95 0076 275 EXIT: TSTB IFB$B_ORGCASE(R10) ; is this sequential f.o. ?
```

```
OE 22 AA 0F 13 0079 276      BEQL  CLRTEF      ; branch if yes
54 6C AA 05 E0 007B 277      BBS   #IFB$V_BIO,IFB$B_FAC(R10); RETURN; return if block i/o accessed
00000000'EF 17 0080 278      MOVL  IFB$L_COCK_BDB(R10),R4 ; restore lock bdb address
                                JMP   RMS$RLSPLG      ; release lock and exit routine
                                ; with rsb which will deallocate
                                ; buffers if allocated before
                                ;
                                ; clear the auto extend truncate at eof flag
                                ;
                                ;
008A 280
008A 281
008A 282
008A 283
008A 284
008A 285 CLRTEF: CSB   #IFB$V_TEF,(R10)
008E 287
008E 288 ;
008E 289 ; all set
008E 290 ;
008E 291
008E 292 RETURN:
05 008E 293 RSB
008F 294 ; call back to alloc_buf
; or return to caller
```



```
008F 296      .SBTTL RMSALLOC_BUF - CO-ROUTINE TO ALLOCATE/DEALLOCATE BUFFERS
008F 297
008F 298      :++
008F 299
008F 300      : RMSALLOC_BUF
008F 301
008F 302      : this routine is called to allocate a buffer and lock bdb for use in extend
008F 303      : processing if no streams connected. the caller is called back so that
008F 304      : control returns to this routine to return the buffer when the caller
008F 305      : does an rsb. if no action was taken on the initial call to this routine,
008F 306      : control will not return here.
008F 307
008F 308      : inputs:
008F 309
008F 310      :         r11      impure area address
008F 311      :         r10      ifab address
008F 312      :         r5       size of buffer desired in blocks
008F 313      :         r6       1 if lock blb needed, 0 otherwise.
008F 314
008F 315      : outputs:
008F 316
008F 317      :         r9       set to ifab addr after initial return if buff allocated
008F 318      :         r1-r6    destroyed
008F 319
008F 320      :         r0       status of buffer allocation if performed, else success
008F 321      :         after initial return
008F 322
008F 323      :
008F 324      :--
008F 325
008F 326      RMSALLOC_BUF::
008F 327      MOVL    #1,R0      ; assume success
008F 328      BBS     #IFBSV_BIO, IFBSB_FAC(R10), 10$ ; always allocate if bio.
008F 329      TSTL    IFBSL_IRAB_LNK(R10) ; any streams connected?
008F 330      BEQL    10$      ; NEQ if so, then just return and
008F 331      RSB     ; no callback will occur.
008F 332      ASHL    #9,R5,R5 ; size of buffer in bytes
008F 333      CLRL    R9       ; don't want bcnt (no irab)
008F 334
008F 335      :
008F 336      : If AI or BI journaling, allocate a bigger buffer. The bigger buffer
008F 337      : contains the buffer (as originally required), a BDB to describe the
008F 338      : journal operation, and an RJR to describe the journal entry.
008F 339      : (For a nice picture of this, see RM3CONN.)
008F 340      :
008F 341      :
008F 342      PUSHL   R5       ; save "real" buffer size
008F 343      BBC     #IFBSV_AI,IFBSB_JNLFLG(R10), 11$ ; skip if not AI jnlng
008F 344      ADDL2   #<RJRSC_BKTLEN+BDBSC_BLN>,R5 ; extra BDB and RJR if AI jnlng
008F 345      BBC     #IFBSV_BI,IFBSB_JNLFLG(R10), 12$ ; skip if not BI journaling
008F 346      ADDL3   #<RJRSC_BKTLEN+BDBSC_BLN>,(SP),R5 ; add in more for BI
008F 347      BSBW   RMSBDBALOC_ALT ; get the buffer
008F 348      POPL    R5       ; restore "real" buffer size
008F 349      MOVL    R10,R9 ; restore ifab addr to r9
008F 350      BLBC    R0,20$ ; on error, cleanup and return
008F 351      TSTB    IFBSB_JNLFLG(R9) ; any journaling?
008F 352      BEQL    13$      ; skip if none

06 22 50 01 D0 008F 327
AA 05 E0 0092 328
1C AA D5 0097 329
01 13 009A 330
05 009C 331
55 55 09 78 009D 332 10$:
59 D4 00A1 333
00A3 334
00A3 335
00A3 336
00A3 337
00A3 338
00A3 339
00A3 340
00A3 341
07 00A0 CA 55 DD 00A3 342
55 00000094 8F C0 00A5 343
08 00A0 CA 02 E1 00AB 344
55 6E 00000094 8F C1 00B2 345 11$:
FF 3D' 30 00C0 346
55 8ED0 00C3 347 12$:
59 5A D0 00C6 348
OF 50 E9 00C9 349
00A0 C9 95 00CC 350
07 13 00D0 351
00D0 352
```

```
54 40 A9 D0 00D2 353      MOVL  IFB$L_BDB_FLNK(R9),R4      ; get BDB address for CARVE_BDB
    00C8 30 00D6 354      BSBW  RM$CARVE_BDB              ; arrange journaling
    9E 16 00D9 355 13$:   JSB    @ (SP)+                  ; call the caller back. when
    00DB 356              ; caller rsb's, come back here.
    50 DD 00DB 357 20$:   PUSHL  R0                        ; Save status.
    FF20' 30 00DD 358      BSBW  RM$RELEASALL              ; Return all buffers, desc., unmap
    00E0 359              ; global section if neccessary.
    50 8ED0 00E0 360      POPL  R0                        ; Restore status.
17 22 AA 05 E1 00E3 361   BBC    #IFB$V_BIO,IFB$B_FAC(R10),40$ ; All done if not block i/o.
    55 5A D0 00E8 362      MOVL  R10, R5                  ; Get ifab addr into R5.
    00EB 363
    00EB 364
    00EB 365      ASSUME  IFB$L_IRAB_LNK EQ IRB$L_IRAB_LNK
    00EB 366
    55 1C A5 D0 00EB 367 30$: MOVL  IFB$L_IRAB_LNK(R5), R5 ; any more streams connected?
    0E 13 00EF 368      BEQL  40$                          ; EQL no streams, just return.
    50 DD 00F1 369      PUSHL  R0                          ; Save status code.
    FFOA' 30 00F3 370      BSBW  RM$ALBDB                  ; Restore BDB for block i/o.
    0084 CA B6 00F6 371      INCW  IFB$W_AVLCL(R10)         ; Bump local buffer count.
    50 8ED0 00FA 372      POPL  R0                          ; Restore status code
    EC 11 00FD 373      BRB    30$                          ; Look for more.
    05 00FF 374 40$:     RSB                                ; And return.
```



```
0100 376 .SBTTL RM$EXTEND_XAB - ROUTINE TO HANDLE ALLOCATION XABS
0100 377
0100 378 :++
0100 379 :
0100 380 RM$EXTEND_XAB
0100 381
0100 382 this routine is called from rm$xab_scan whenever an allocation xab
0100 383 is found on an $extend.
0100 384
0100 385 the xab is checked for validity and its parameters are used to extend
0100 386 the file. if there are no errors, this routine returns to continue
0100 387 the xab scan.
0100 388
0100 389 inputs:
0100 390
0100 391 @sp return address if successful
0100 392 ap rm$xab_scan argument list addr
0100 393 r11 impure area addr
0100 394 r10 ifab addr
0100 395 r9 ifab addr
0100 396 r8 fab addr
0100 397 r4 zero if this is the first call, else the area i.d. of
0100 398 the last area + 1
0100 399 r3 xab addr
0100 400
0100 401 the xab has already been checked for basic validity
0100 402
0100 403 outputs:
0100 404
0100 405 r4 this xab's area i.d. + 1
0100 406 r0-r2,r5-r6 destroyed
0100 407
0100 408 if an error occurs, the return address is popped from the stack,
0100 409 r0 is set to the error code, and an rsb is performed back to caller
0100 410 of rm$xab_scan.
0100 411
0100 412 :--
0100 413
0100 414 RM$EXTEND_XAB::
0100 415 CMPB #IFB$C_IDX,IFB$B_ORGCASE(R10); if index file
0100 416 BNEQ 3$
0100 417 BBSS #0,R4,2$
0100 418 2$: BRB 5$
0100 419 3$: BBSS #0,R4,ERRIMX ; branch if duplicate xab
0100 420
0100 421 :
0100 422 : !!! note - different test for
0100 423 : indexed f.o. required !!!
0100 424 :
0100 425 :
0100 426 TSTB XAB$B_AID(R3) ; must be area 0
0100 427 BNEQ ERRAID ; branch if not
0100 428 5$: MOVL R3,R6 ; xab addr to right reg
0100 429 BSBB XTND ; go perform extension
0100 430 BLBS R0,10$ ; branch on success
0100 431 POPL R1 ; pop return pc
0100 432 10$: RSB
```

23	AA	02	91	0100	415				
		06	12	0104	416				
00	54	00	E2	0106	417				
		09	11	010A	418	2\$:			
11	54	00	E2	010C	419	3\$:			
				0110	420				
				0110	421				
				0110	422				
				0110	423				
				0110	424				
				0110	425				
17	A3	95		0110	426				
	0F	12		0113	427				
56	53	D0		0115	428	5\$:			
	0D	10		0118	429				
03	50	E8		011A	430				
	51	8ED0		011D	431				
		05		0120	432	10\$:			

RMSOEXTEN
V04-000

DISPATCH FOR EXTEND OPERATION L 14
RM\$EXTEND_XAB - ROUTINE TO HANDLE ALLOCA 16-SEP-1984 01:18:09 VAX/VMS Macro V04-00
5-SEP-1984 16:24:55 [RMS.SRC]RMSOEXTEN.MAR;1

Page 11
(8)

```
0121 433
0121 434 ;++
0121 435 ;
0121 436 ; handle errors
0121 437 ;
0121 438 ;--
0121 439
FEDC' 31 0121 440 ERRIMX: BRW RM$ERRIMX ; duplicate alloc. xab
0124 441
FED9' 31 0124 442 ERRRAID: BRW RM$ERRRAID ; bad area i.d.
```



```
0127 444  
0127 445 :++  
0127 446 :  
0127 447 xtnd subroutine to perform the extend.  
0127 448 :  
0127 449 inputs:  
0127 450 :  
0127 451 r11 impure area addr  
0127 452 r10 ifab addr  
0127 453 r9 ifab addr  
0127 454 r8 fab addr  
0127 455 r6 xab addr if any, else 0  
0127 456 r3 xab or fab addr  
0127 457 :  
0127 458 outputs:  
0127 459 :  
0127 460 r0 status code  
0127 461 xab$l_alq or  
0127 462 fab$l_alq # of blocks extended, 0 if none  
0127 463 fab$l_stv fab$l_stv + # of blocks extended  
0127 464 r1,r2,r5,r6 destroyed  
0127 465 :  
0127 466 :--  
0127 467 :  
1018 8F BB 0127 468 XTND: PUSHR #^M<R3,R4,AP>  
0128 469 :  
0128 470 ASSUME XAB$L_ALQ EQ FAB$L_ALQ  
0128 471 :  
55 10 A3 D0 0128 472 MOVL XAB$L_ALQ(R3),R5 ; set extend size  
69 15 012F 473 BLEQ ERRALQ ; branch if bad  
10 A3 D4 0131 474 CLRL XAB$L_ALQ(R3) ; and initialize actual extend size  
OE 22 AA 05 E0 0134 475 BBS #IFB$V_BIO,IFB$B_FAC(R10),10$  
23 AA 02 91 0139 476 CMPB #IFB$C_IDX,IFB$B_ORGCASE(R10)  
08 12 013D 477 BNEQ 10$  
00000000'EF 16 013F 478 JSB RM$EXTEND3  
4E 11 0145 479 BRB XTNDXIT  
FEB6' 30 0147 480 10$: BSBW RM$EXTEND0 ; do the extend  
48 50 E9 014A 481 BLBC R0,XTNDXIT ; branch on failure  
014D 482 :  
014D 483 :  
014D 484 : note: r1 = start vbn of extent  
014D 485 : r6 = end vbn of extent + 1  
014D 486 :  
014D 487 :  
10 A3 53 6E D0 014D 488 MOVL (SP),R3 ; restore xab/fab address  
56 51 C3 0150 489 SUBL3 R1,R6,FAB$L_ALQ(R3) ; calculate extend size  
OC A8 10 A3 C0 0155 490 ADDL2 FAB$L_ALQ(R3),FAB$L_STV(R8); and add it in to stv for total  
31 22 AA 05 E0 015A 491 BBS #IFB$V_BIO,IFB$B_FAC(R10),SEQEXT; branch if block i/o accessed  
015F 492 CASE TYPE=B,SRC=IFB$B_ORGCASE(R10),-  
015F 493 DISPLIST=<SEQEXT,REEXT> ; dispatch based on file org  
26 11 0168 494 BRB SEQEXT ; treat like sequential
```

```
016A 496
016A 497 :++
016A 498 :
016A 499 : relative file extend - format the buckets
016A 500 :
016A 501 :--
016A 502 :
016A 503 RELEXT:
11 6A 38 E0 016A 504 BBS #IFB$V_SEQFIL,(R10),SEQSHR ; branch if seq file shr'd
00000000'EF 16 016E 505 JSB RM$FMT_BKT2 ; go format the buckets
00000000'EF 16 0174 506 JSB RM$UPD_PROLOG2 ; and update the prolog
6C AA D4 017A 507 X: CLRL IFB$L_LOCK_BDB(R10) ; say lock bdb gone
16 11 017D 508 BRB XTNDXIT
017F 509 SEQSHR:
54 6C AA D0 017F 510 MOVL IFB$L_LOCK_BDB(R10),R4 ; set up r4 to release lock bdb
50 DD 0183 511 PUSHL R0 ; save status
00000000'EF 16 0185 512 JSB RM$SETHEBK ; set hbk and release lock on -1
50 8ED0 018B 513 POPL R0 ; restore status
EA 11 018E 514 BRB X ; and exit
0190 515 :
0190 516 :++
0190 517 :
0190 518 : sequential file or block i/o extend - update high block
0190 519 :
0190 520 :--
0190 521 :
70 AA 56 01 C3 0190 522 SEQEXT: SUBL3 #1,R6,IFB$L_HBK(R10) ; set new high block
0195 523 :
0195 524 XTNDXIT:
1018 8F BA 0195 525 POPR #^M<R3,R4,AP> ; restore regs
05 0199 526 RSB
019A 527 :
019A 528 :++
019A 529 :
019A 530 : handle invalid alq value error
019A 531 :
019A 532 :--
019A 533 :
019A 534 ERRALQ:
F4 11 019A 535 RMSERR ALQ
019F 536 BRB XTNDXIT
```



```
01A1 538 .SUBTITLE RM$CARVE_BDB
01A1 539 :++
01A1 540 : RM$CARVE_BDB
01A1 541 :
01A1 542 : Carve up the bigger buffer into a BDB, RJR, and the original buffer.
01A1 543 :
01A1 544 : Inputs:
01A1 545 :
01A1 546 :     R4     points to real BDB.
01A1 547 :     R5     has the "real" buffer size.
01A1 548 :     R9     has the IFAB address.
01A1 549 :
01A1 550 : Outputs:
01A1 551 :
01A1 552 :     BDBs initied.
01A1 553 :
01A1 554 :--
01A1 555
01A1 556 RM$CARVE_BDB:
01A1 557
01A1 558 ; R6 will point to the BDB used for the AI journal entry.
01A1 559
29 00A0 C9 03 D0 01A1 560      MOVL    BDB$L_ADDR(R4),R6      ; get buffer address
      56 18 A4 03 E1 01A5 561      BBC     #IFB$V_AI,IFB$B_JNLFLG(R9),10$ ; skip if no AI jnling
      34 A4 56 D0 01AB 562      MOVL    R6,BDB$L_AI_BDB(R4)      ; put AI_BDB address in real BDB
01AF 563
01AF 564 :
01AF 565 : Initialize the AI_BDB
01AF 566 :
01AF 567
      08 A6 0C 90 01AF 568      MOVB    #BDB$C_BID,BDB$B_BID(R6)      ; block ID
      09 A6 14 90 01B3 569      MOVB    #<BDB$C_BLN/4>,BDB$B_BLN(R6) ; and block length
      66 56 D0 01B7 570      MOVL    R6,BDB$C_FLINK(R6)      ; bdb queue is null
      04 A6 56 D0 01BA 571      MOVL    R6,BDB$L_BLINK(R6)
16 A6 55 0044 8F A1 01BE 572      ADDW3   #RJR$C_BKTLEN,R5,BDB$W_SIZE(R6) ; size = RJR + "real" buffer
      18 A6 50 A6 9E 01C5 573      MOVAB   BDB$C_BLN(R6),BDB$L_ADDR(R6) ; buffer address
01CA 574
01CA 575 :
01CA 576 : Now, correct the "real" BDB's buffer address to point past AI_BDB and RJR.
01CA 577 :
01CA 578
18 A4 18 A6 00000044 8F C1 01CA 579      ADDL3   #RJR$C_BKTLEN,BDB$L_ADDR(R6),BDB$L_ADDR(R4)
      24 00A0 C9 02 E1 01D4 580 10$:      BBC     #IFB$V_BI,IFB$B_JNLFLG(R9),20$ ; skip if not BI jnling
01DA 581
01DA 582 :
01DA 583 : Carve out and initialize the BI_BDB.
01DA 584 :
01DA 585 :
01DA 586 : First, calculate BI_BDB address. The BI_BDB is allocated after the
01DA 587 : actual buffer. The actual buffer is pointed to by BDB$L_ADDR(R4).
01DA 588 : The actual buffer size is in R5.
01DA 589 :
01DA 590 : R6 will point to the BDB used for the BI journal entry.
01DA 591
56 18 A4 55 C1 01DA 592      ADDL3   R5,BDB$L_ADDR(R4),R6      ; pointer to BI_BDB
      30 A4 56 D0 01DF 593      MOVL    R6,BDB$L_BI_BDB(R4)      ; filled in "real" BDB
      08 A6 0C 90 01E3 594      MOVB    #BDB$C_BID,BDB$B_BID(R6)      ; block ID
```


16 A6	09 A6 14	90 01E7 595	MOVB	#<BDB\$C_BLN/4>,BDB\$B_BLN(R6)	; and block length
	66 56	D0 01EB 596	MOVL	R6,BDB\$C_FLINK(R6)	; bdb queue is null
	04 A6 56	D0 01EE 597	MOVL	R6,BDB\$L_BLINK(R6)	
	55 0044 8F	A1 01F2 598	ADDW3	#RJR\$C_BRTLEN,R5,BDB\$W_SIZE(R6)	; buffer size = RJR + buffer
18 A6 50 A6		9E 01F9 599	MOVAB	BDB\$C_BLN(R6),BDB\$L_ADDR(R6)	; buffer address
		05 01FE 600			
		01FF 601 20\$:	RSB		
		01FF 602			
		01FF 603	.END		

RMSOEXTEN
Symbol table

DISPATCH FOR EXTEND OPERATION

D 15

16-SEP-1984 01:18:09 VAX/VMS Macro V04-00
5-SEP-1984 16:24:55 [RMS.SRC]RMSOEXTEN.MAR;1

Page 16
(13)

\$\$PSECT_EP	=	00000000		
\$\$TMP	=	00000000		
\$\$RMSTEST	=	0000001A		
\$\$RMS_PBUGCHK	=	00000010		
\$\$RMS_TBUGCHK	=	00000008		
\$\$RMS_UMODE	=	00000004		
BDB\$B_BID	=	00000008		
BDB\$B_BLN	=	00000009		
BDB\$C_BID	=	0000000C		
BDB\$C_BLN	=	00000050		
BDB\$S_ADDR	=	00000018		
BDB\$S_AI_BDB	=	00000034		
BDB\$S_BI_BDB	=	00000030		
BDB\$S_BLINK	=	00000004		
BDB\$S_FLINK	=	00000000		
BDB\$W_SIZE	=	00000016		
CLRTEF		0000008A	R	01
CSH\$M_LOCK	=	00000001		
CSH\$M_NOBUFFER	=	00000008		
CSH\$M_NOREAD	=	00000004		
DEV\$V_RND	=	0000001C		
ERRAID		00000124	R	01
ERRALQ		0000019A	R	01
ERRFAC		00000007	R	01
ERRIMX		00000121	R	01
ERRIOP		00000004	R	01
EXIT		00000076	R	01
EXTEND_IT		0000002A	R	01
FAB\$S_ALQ	=	00000010		
FAB\$S_FOP	=	00000004		
FAB\$S_STV	=	0000000C		
FOP	=	00000020		
GETXAB		00000060	R	01
IFB\$B_BKS	=	0000005E		
IFB\$B_FAC	=	00000022		
IFB\$B_JNLFLG	=	000000A0		
IFB\$B_ORGCASE	=	00000023		
IFB\$C_IDX	=	00000002		
IFB\$C_REL	=	00000001		
IFB\$C_SEQ	=	00000000		
IFB\$S_BDB_FLNK	=	00000040		
IFB\$S_HBK	=	00000070		
IFB\$S_IRAB_LNK	=	0000001C		
IFB\$S_LOCK_BDB	=	0000006C		
IFB\$S_PRIM_DEV	=	00000000		
IFB\$V_AI	=	00000003		
IFB\$V_BI	=	00000002		
IFB\$V_BIO	=	00000005		
IFB\$V_DAP	=	0000003E		
IFB\$V_SEQFIL	=	00000038		
IFB\$V_TEF	=	00000036		
IFB\$V_WRTACC	=	00000030		
IFB\$W_AVI_CL	=	00000084		
IRB\$S_IRAB_LNK	=	0000001C		
NT\$EXTEND		*****	X	01
PIOSA_TRACE		*****	X	01
RELEXT		0000016A	R	01

RETURN		0000008E	R	01
RJR\$C_BKTLEN	=	00000044		
RMSALBDB		*****	X	01
RMSALLOC_BUF		0000008F	RG	01
RMSBDBALOC_ALT		*****	X	01
RMSCACHE		*****	X	01
RMSCARVE_BDB		000001A1	R	01
RMSERRAID		*****	X	01
RMSERRFAC		*****	X	01
RMSERRIMX		*****	X	01
RMSERRIOP		*****	X	01
RMSEX RMS		*****	X	01
RMSEXTEND0		*****	X	01
RMSEXTEND3		*****	X	01
RMSEXTEND_XAB		00000100	RG	01
RMSFMT_BKT2		*****	X	01
RMSFSET		*****	X	01
RMSLOCK_PROLOG		*****	X	01
RMSRELEASESALL		*****	X	01
RMSRLSPLG		*****	X	01
RMSSETHEBK		*****	X	01
RMSUPD_PROLOG2		*****	X	01
RMSXAB_SCAN		*****	X	01
RMS\$EXTEND	=	00000008	RG	01
RMS\$ALQ	=	00018404		
SEQEXT		00000190	R	01
SEQSHR		0000017F	R	01
TPT\$S_EXTEND		*****	X	01
X		0000017A	R	01
XAB\$B_AID	=	00000017		
XAB\$C_ALL	=	00000014		
XAB\$C_ALLLEN	=	00000020		
XAB\$S_ALQ	=	00000010		
XAB_ARGS		00000000	R	01
XBC\$C_EXTALL		*****	X	01
XTND		00000127	R	01
XTNDXIT		00000195	R	01

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

PSECT name	Allocation	PSECT No.	Attributes
ABS	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
RMSRMS	000001FF (511.)	01 (1.)	PIC USR CON REL GBL NOSHR EXE RD NOWRT NOVEC BYTE
SABSS	00000000 (0.)	02 (2.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE

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

Phase	Page faults	CPU Time	Elapsed Time
Initialization	39	00:00:00.09	00:00:00.44
Command processing	148	00:00:00.72	00:00:05.40
Pass 1	334	00:00:11.54	00:00:29.33
Symbol table sort	0	00:00:01.53	00:00:01.87
Pass 2	112	00:00:02.57	00:00:05.86
Symbol table output	11	00:00:00.10	00:00:00.21
Psect synopsis output	3	00:00:00.03	00:00:00.03
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	649	00:00:16.60	00:00:43.18

The working set limit was 1650 pages.

63802 bytes (125 pages) of virtual memory were used to buffer the intermediate code.

There were 60 pages of symbol table space allocated to hold 1176 non-local and 20 local symbols.

603 source lines were read in Pass 1, producing 14 object records in Pass 2.

29 pages of virtual memory were used to define 28 macros.

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

Macro library name	Macros defined
_\$255\$DUA28:[RMS.OBJ]RMS.MLB;1	18
_\$255\$DUA28:[SYS.OBJ]LIB.MLB;1	1
_\$255\$DUA28:[SYSLIB]STARLET.MLB;2	5
TOTALS (all libraries)	24

1323 GETs were required to define 24 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LIS\$:RMSOEXTEN/OBJ=OBJ\$:RMSOEXTEN MSRC\$:RMSOEXTEN/UPDATE=(ENH\$:RMSOEXTEN)+EXECML\$/LIB+LIB\$:RMS/LIB

0329 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

