



```
EEEEEEEEEE XX XX CCCCCCCC RRRRRRRR TTTTTTTTTT AAAAAA CCCCCCCC PPPPPPPP
EEEEEEEEEE XX XX CCCCCCCC RRRRRRRR TTTTTTTTTT AAAAAA CCCCCCCC PPPPPPPP
EE XX XX CC CCCCCCCC RRRRRRRR RR RR TTTTTTTTTT AA AA CC CCCCCCCC PP PP
EE XX XX CC CCCCCCCC RRRRRRRR RR RR TTTTTTTTTT AA AA CC CCCCCCCC PP PP
EE XX XX CC CCCCCCCC RRRRRRRR RR RR TTTTTTTTTT AA AA CC CCCCCCCC PP PP
EE XX XX CC CCCCCCCC RRRRRRRR RR RR TTTTTTTTTT AA AA CC CCCCCCCC PP PP
EEEEEEEEEE XX XX CCCCCCCC RRRRRRRR TTTTTTTTTT AAAAAA CCCCCCCC PPPPPPPP
EEEEEEEEEE XX XX CCCCCCCC RRRRRRRR TTTTTTTTTT AAAAAA CCCCCCCC PPPPPPPP
EE XX XX CC CCCCCCCC RRRRRRRR RR RR TTTTTTTTTT AA AA CC CCCCCCCC PP PP
EE XX XX CC CCCCCCCC RRRRRRRR RR RR TTTTTTTTTT AA AA CC CCCCCCCC PP PP
EE XX XX CC CCCCCCCC RRRRRRRR RR RR TTTTTTTTTT AA AA CC CCCCCCCC PP PP
EEEEEEEEEE XX XX CCCCCCCC RRRRRRRR TTTTTTTTTT AAAAAA CCCCCCCC PPPPPPPP
EEEEEEEEEE XX XX CCCCCCCC RRRRRRRR TTTTTTTTTT AAAAAA CCCCCCCC PPPPPPPP
```

```
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 IIIIII SSSSSSSS
LLLLLLLLLL IIIIII SSSSSSSS
LLLLLLLLLL IIIIII SSSSSSSS
```

```

1 0001 0 MODULE  exch$rtacp                %TITLE 'RT11 directory routines'
2 0002 0
3 0003 0          IDENT = 'V04-000'
4 0004 0          ADDRESSING_MODE (EXTERNAL=LONG_RELATIVE, NONEXTERNAL=WORD_RELATIVE)
5 0005 0          ) =
6 0006 1 BEGIN
7 0007 1
8 0008 1 *****
9 0009 1 *
10 0010 1 *  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
11 0011 1 *  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
12 0012 1 *  ALL RIGHTS RESERVED.
13 0013 1 *
14 0014 1 *  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
15 0015 1 *  ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
16 0016 1 *  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
17 0017 1 *  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
18 0018 1 *  OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
19 0019 1 *  TRANSFERRED.
20 0020 1 *
21 0021 1 *  THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
22 0022 1 *  AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
23 0023 1 *  CORPORATION.
24 0024 1 *
25 0025 1 *  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
26 0026 1 *  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
27 0027 1 *
28 0028 1 *
29 0029 1 *****
30 0030 1
31 0031 1 ++
32 0032 1 FACILITY:    EXCHANGE - Foreign volume interchange facility
33 0033 1
34 0034 1 ABSTRACT:    RT-11 directory manipulation routines
35 0035 1
36 0036 1 ENVIRONMENT:  VAX/VMS User mode
37 0037 1
38 0038 1 AUTHOR:      CW Hobbs          CREATION DATE: 29-Nov-1982
39 0039 1
40 0040 1 MODIFIED BY:
41 0041 1
42 0042 1          V03-002 CWH3002          CW Hobbs          12-Apr-1984
43 0043 1          If the directory shows a device larger than the actual
44 0044 1          device, then print a warning and write lock the volume.
45 0045 1
46 0046 1
47 0047 1 --
48 0048 1
49 0049 1 Include files:
50 0050 1
51 0051 1 MACRO $module_name_string = 'exch$rtacp' %;          ! The require file needs to know our module name
52 0052 1 REQUIRE 'SRCS:EXCREQ'                                ! Facility-wide require file
53 0053 1

```

```
55 0150 1 %SBTTL 'Module table of contents'
56 0151 1
57 0152 1 ! Module table of contents:
58 0153 1 !
59 0154 1 FORWARD ROUTINE
60 0155 1     exch$rtacp_check_position : NOVALUE, ! Find directory entry if it has moved
61 0156 1     exch$rtacp_clean_directory : NOVALUE, ! Shuffle and/or split directories as needed
62 0157 1     exch$rtacp_consolidate, ! Compress directory structure
63 0158 1     exch$rtacp_find_empty_area, ! Find free space
64 0159 1     exch$rtacp_find_file, ! RT-11 directory search routine
65 0160 1     exch$rtacp_next_entry, ! Retrieve next entry from RT-11 directory
66 0161 1     exch$rtacp_verify_directory ! Verify directory structure and compute volume size
67 0162 1 ;
68 0163 1
69 0164 1 ! EXCHANGE facility routines
70 0165 1 !
71 0166 1 EXTERNAL ROUTINE
72 0167 1     exch$cmd_fetch_recfmt_implied : NOVALUE, ! Get or assume the value for /RECORD_FORMAT
73 0168 1     exch$cmd_match_filename, ! Compare expanded file names for match
74 0169 1     exch$io_rt11_read, ! Read blocks from a random access device
75 0170 1     exch$io_rt11_write, ! Write blocks to a random access device
76 0171 1     exch$pdp_flush_write_buffer, ! Flush any records waiting in the write buffer
77 0172 1     exch$pdp_get, ! Get functions for small PDP record structure
78 0173 1     exch$pdp_put, ! Put functions for small PDP record structure
79 0174 1     exch$rt11_dirseg_flush, ! Write out directory segments
80 0175 1     exch$rt11_dirseg_get, ! Return pointer to specific directory segment
81 0176 1     exch$rt11_dirseg_get_nochk : jsb_rlr2, ! Return pointer to directory segment without checking
82 0177 1     exch$rt11_dirseg_put, ! Write a specific directory segment
83 0178 1     exch$rt11_expand_filename, ! Convert directory entry to ASCII filename
84 0179 1     exch$util_fao_buffer, ! Do an FAO conversion
85 0180 1     exch$util_radix50_from_ascii, ! Convert characters to Radix-50 from Ascii
86 0181 1     exch$util_radix50_to_ascii, ! Convert characters from Radix-50 to Ascii
87 0182 1     exch$util_rt11ctx_allocate, ! Get an RT-11 context block
88 0183 1     exch$util_rt11ctx_release : NOVALUE, ! Give it back
89 0184 1     exch$util_vm_allocate, ! Get some virtual memory
90 0185 1     exch$util_vm_allocate_zeroed, ! Get some virtual memory, initialized to zero
91 0186 1     exch$util_vm_release ! Return some virtual memory
92 0187 1 ;
93 0188 1
94 0189 1 ! Equated symbols:
95 0190 1 !
96 0191 1 ! LITERAL
97 0192 1 ! ;
98 0193 1 !
99 0194 1 ! Bound declarations:
100 0195 1 !
101 0196 1 ! BIND
102 0197 1 ! ;
```

```
104 0198 1 GLOBAL ROUTINE exch$rtacp_check_position (ctx : $ref_bblock) : NOVALUE = %SBTTL 'exch$rtacp_check_pos
105 0199 2 BEGIN
106 0200 2 ++
107 0201 2
108 0202 2 FUNCTIONAL DESCRIPTION:
109 0203 2
110 0204 2 Check that the directory entry described in the context block is at the correct position. Several e
111 0205 2 might have caused the directory to be rearranged, such as the creation of a file since the context w
112 0206 2 found. If the entry has moved, find it and set the pointers to the new position.
113 0207 2
114 0208 2 INPUT/OUTPUT:
115 0209 2
116 0210 2 ctx - pointer to block describing the directory context
117 0211 2
118 0212 2 IMPLICIT INPUTS:
119 0213 2
120 0214 2 none
121 0215 2
122 0216 2 IMPLICIT OUTPUTS:
123 0217 2
124 0218 2 none
125 0219 2
126 0220 2 ROUTINE VALUE:
127 0221 2
128 0222 2 none
129 0223 2
130 0224 2 SIDE EFFECTS:
131 0225 2
132 0226 2 none
133 0227 2 --
134 0228 2
135 0229 2 $dbgtrc_prefix ('exch$rtacp_check_position> ');
136 0230 2
137 0231 2 LOCAL
138 0232 2 seg : $ref_bblock, ! a pointer to the current directory segment
139 0233 2 ctx2 : $ref_bblock, ! a pointer to a second context in case we need to scan
140 0234 2 flags
141 0235 2 ;
142 0236 2
143 0237 2 BIND
144 0238 2 volb = ctx [rt11ctx$a_assoc_volb] : $ref_bblock
145 0239 2 ;
146 0240 2
147 0241 2 $block_check (2, .ctx, rt11ctx, 443);
148 0242 2 $block_check (2, .volb, volb, 445);
149 0243 2 $logic_check (5, (exch$rtacp_verify_directory (.volb)), 182);
150 0244 2
151 0245 2
152 0246 2 ! We need to check to make sure that the number of segments has not been lowered. If so, our entry address
153 0247 2 ! be sitting in the hyperspace beyond the segments in use. Hence, grab a pointer to the root segment.
154 0248 2
155 0249 2 seg = exch$rt11_dirseg_get_nochk (.volb, 1);
156 0250 2 $logic_check (2, (.seg_NEQ 0), 147);
```

```
158 0251 2 ! Check to see if the entry position info is correct, if so we can simply return
159 0252 2
160 0253 2 IF .ctx [rt11ctx$L_seg_number] LEQU .seg [rt11hdr$w_high_seg]
161 0254 2 AND
162 0255 2 CH$EQL (rt11ctx$s_entry, ctx [rt11ctx$t_entry], rt11ctx$s_entry, .ctx [rt11ctx$a_ent_address])
163 0256 2 THEN
164 0257 2 RETURN;
165 0258 2
166 0259 2 $trace_print_lit ('looking for new position');
167 0260 2
168 0261 2 ! Get a context block for the search
169 0262 2
170 0263 2 ctx2 = exch$util_rt11ctx_allocate (.volb, 0);
171 0264 2
172 0265 2 ! Loop through all the segments in the directory looking for this file entry
173 0266 2
174 0267 2 flags = rtnxt$m_permanent OR rtnxt$m_empty OR rtnxt$m_tentative OR rtnxt$m_skip_check OR rtnxt$m_skip_expand
175 0268 2 WHILE exch$rtacp_next_entry (.ctx2, :flags) NEQ 0
176 0269 2 DO
177 0270 3 BEGIN
178 0271 3
179 0272 3 IF CH$EQL (rt11ctx$s_entry, ctx [rt11ctx$t_entry], rt11ctx$s_entry, ctx2 [rt11ctx$t_entry])
180 0273 3 THEN
181 0274 4 BEGIN
182 0275 4
183 0276 4 $trace_print_fao ('Relocated "!AF"', .ctx [rt11ctx$L_exp_fullname_len], ctx [rt11ctx$t_exp_fullname])
184 0277 4
185 0278 4 ! if the file has moved we are in deep trouble
186 0279 4
187 0280 4 $logic_check (3, (.ctx [rt11ctx$L_start_block] EQL .ctx2 [rt11ctx$L_start_block]), 120);
188 0281 4
189 0282 4 ! Put the new position info into the context block
190 0283 4
191 0284 4 ctx [rt11ctx$L_seg_number] = .ctx2 [rt11ctx$L_seg_number];
192 0285 4 ctx [rt11ctx$a_seg_address] = .ctx2 [rt11ctx$a_seg_address];
193 0286 4 ctx [rt11ctx$a_ent_address] = .ctx2 [rt11ctx$a_ent_address];
194 0287 4
195 0288 4 ! Give the extra context back
196 0289 4
197 0290 4 exch$util_rt11ctx_release (.ctx2);
198 0291 4
199 0292 4 RETURN;
200 0293 3 END;
201 0294 3
202 0295 2 END;
203 0296 2
204 0297 2 $logic_check (0, (false), 121); ! We should have found it
205 0298 2
206 0299 2 RETURN;
207 0300 1 END;
```

```
.TITLE EXCH$RTACP RT11 directory routines
.IDENT \V04-000\

.EXTRN EXCH$CMD_FETCH_RECfmt IMPLIED
.EXTRN EXCH$CMD_MATCH_FILENAME
```

```

.EXTRN EXCH$IO_RT11_READ
.EXTRN EXCH$IO_RT11_WRITE
.EXTRN EXCH$PDP_FLUSH_WRITE_BUFFER
.EXTRN EXCH$PDP_GET, EXCH$PDP_PUT
.EXTRN EXCH$RT11_DIRSEG_FLUSH
.EXTRN EXCH$RT11_DIRSEG_GET
.EXTRN EXCH$RT11_DIRSEG_GET_NOCHK
.EXTRN EXCH$RT11_DIRSEG_PUT
.EXTRN EXCH$RT11_EXPAND_FILENAME
.EXTRN EXCH$UTIL_FAO_BUFFER
.EXTRN EXCH$UTIL_RADIX50_FROM_ASCII
.EXTRN EXCH$UTIL_RADIX50_TO_ASCII
.EXTRN EXCH$UTIL_RT11CTX_ALLOCATE
.EXTRN EXCH$UTIL_RT11CTX_RELEASE
.EXTRN EXCH$UTIL_VM_ALLOCATE
.EXTRN EXCH$UTIL_VM_ALLOCATE_ZEROED
.EXTRN EXCH$UTIL_VM_RELEASE
.EXTRN EXCH$UTIL_BLOCK_CHECK
.EXTRN EXCH$_BADLOGIC

```

.PSECT EXCH\$RTACP\_CODE, NOWRT, 2

```

.ENTRY EXCH$RTACP_CHECK_POSITION, Save R2,R3,R4,- : 0198
      R5,R6,R7,R8,R9
      MOVAB EXCH$UTIL_BLOCK_CHECK, R9
      MOVAB LIB$STOP, R8
      MOVL #EXCH$_BADLOGIC, R7
      MOVL CTX, R4 : 0238
      MOVL #8519924, R2 : 0241
      MOVZWL #443, R1
      MOVL R4, R0
      JSB EXCH$UTIL_BLOCK_CHECK
      MOVL #68878579, R2 : 0242
      MOVZWL #445, R1
      MOVL 20(R4), R0
      JSB EXCH$UTIL_BLOCK_CHECK
      MOVL #1, R2 : 0249
      MOVL 20(R4), R1
      JSB EXCH$RT11_DIRSEG_GET_NOCHK
      MOVL R0, SEC
      BNEQ 1$ : 0250
      MOVZBL #147, -(SP)
      PUSHL #1
      PUSHL R7
      CALLS #3, LIB$STOP
      CMPZV #0, #16, 4(SEG), 118(R4) : 0253
      BLSSU 2$
      CMPC3 #14, 56(R4), @126(R4) : 0255
      BEQL 5$
      CLRL -(SP) : 0263
      PUSHL 20(R4)
      CALLS #2, EXCH$UTIL_RT11CTX_ALLOCATE
      MOVL R0, CTX2
      MOVL #55, FLAGS : 0267
      MOVQ CTX2, -(SP) : 0268
      CALLS #2, EXCH$RTACP_NEXT_ENTRY
      TSTL R0

```

03FC 00000

```

59 00000000G EF 9E 00002
58 00000000G 00 9E 00009
57 00000000G 8F D0 00010
54 04 AC D0 00017
52 008200F4 8F D0 0001B
51 01BB 8F 3C 00022
50 54 D0 00027
69 16 0002A
52 041B00F3 8F D0 0002C
51 01BD 8F 3C 00033
50 14 A4 D0 00038
69 16 0003C
52 01 D0 0003E
51 14 A4 D0 00041
00000000G EF 16 00045
52 50 D0 0004B
7E 93 0B 12 0004E
01 DD 00054
57 DD 00056
68 03 FB 00058
10 00 ED 0005B 1$:
08 1F 00062
7E B4 38 A4 0E 29 00064
45 13 0006A
7E D4 0006C 2$:
14 A4 DD 0006E
09000000G EF 02 FB 00071
55 50 D0 00078
56 37 D0 0007B
7E 55 7D 0007E 3$:
0000V CF 02 FB 00081
50 D5 00086

```

76 A4 04 A2  
7E B4 38 A4

09000000G

0000V

38	A5	38	A4	1C	13	00088	BEQL	4\$	:			
				0E	29	0008A	CMPC3	#14, 56(R4), 56(CTX2)	:	0272		
				EC	12	00090	BNEQ	3\$	:			
		76	A4	76	A5	7D	00092	MOVQ	118(CTX2), 118(R4)	:	0284	
		7E	A4	7E	A5	DO	00097	MOVL	126(CTX2), 126(R4)	:	0286	
				55	DD	0009C	PUSHL	CTX2	:	0290		
00000000G				01	FB	0009E	CALLS	#1, EXCH\$UTIL_RT11CTX_RELEASE	:			
						04	000A5	RET	:	0280		
		7E		79	8F	9A	000A6	4\$:	MOVZBL	#121, -(SP)	:	0297
					01	DD	000AA		PUSHL	#1	:	
					57	DD	000AC		PUSHL	R7	:	
		68			03	FB	000AE		CALLS	#3, LIB\$STOP	:	
						04	000B1	5\$:	RET	:	0300	

; Routine Size: 178 bytes, Routine Base: EXCH\$RTACP\_CODE + 0000



```

: 209 0301 1 GLOBAL ROUTINE exch$rtacp_clean_directory (volb : $ref_bblock, %SBTTL 'exch$rtacp_clean_directory (volb, en
: 210 0302 1 ent_cnt, ent_len) : NOVALUE =
: 211 0303 BEGIN
: 212 0304 ++
: 213 0305
: 214 0306 FUNCTIONAL DESCRIPTION:
: 215 0307
: 216 0308 This routine is used like the RT-11 segment split logic, i.e. it makes room for a directory entry.
: 217 0309 of a simple segment split, it may restructure the entire directory. It will add or remove segments
: 218 0310 the chain to produce a directory with approximately the same number of active entries in each segmen
: 219 0311
: 220 0312 This routine assumes that it has been called from exch$rtacp_consolidate. It assumes that every dir
: 221 0313 segment contains only valid directory entries and is terminated by an end-segment marker. It does n
: 222 0314 require that unnecessary entries (see exch$rtacp_consolidate) be removed.
: 223 0315
: 224 0316 INPUTS:
: 225 0317
: 226 0318 volb - pointer to volb which has been connected to the RT-11 device
: 227 0319 ent_cnt - number of entries in the directory
: 228 0320 ent_len - length of a single entry, including extra bytes
: 229 0321
: 230 0322 IMPLICIT INPUTS:
: 231 0323
: 232 0324 none
: 233 0325
: 234 0326 OUTPUTS:
: 235 0327
: 236 0328 none
: 237 0329
: 238 0330 IMPLICIT OUTPUTS:
: 239 0331
: 240 0332 none
: 241 0333
: 242 0334 ROUTINE VALUE:
: 243 0335
: 244 0336 none
: 245 0337
: 246 0338 SIDE EFFECTS:
: 247 0339
: 248 0340 all error conditions are fatal
: 249 0341 --
: 250 0342
: 251 0343 $dbgtrc_prefix ('exch$rtacp_clean_directory> ');
: 252 0344
: 253 0345 LOCAL
: 254 0346 seg : $ref_bblock, ! a pointer to the current directory segment
: 255 0347 cur : $ref_bblock, ! a pointer to the current directory entry
: 256 0348 dir2 : $ref_bblock, ! a pointer to an area where we will assemble a clean direct
: 257 0349 seg2 : $ref_bblock, ! a pointer to the current segment in the clean directory ar
: 258 0350 cur2 : $ref_bblock, ! a pointer to the current entry in the clean directory area
: 259 0351 ctx : $ref_bblock, ! a pointer a context block for looping through the dirty di
: 260 0352 ent2_cnt, ! count of entries in the current clean segment
: 261 0353 seg2_num, ! current segment number in the clean directory
: 262 0354 seg_cnt, ! number of segments for the cleaned directory
: 263 0355 ent_per_seg, ! goal for number of entries per segment
: 264 0356 max_ent_per_seg, ! max number of entries per segment
: 265 0357 flags ! local for our flags to exch$rtacp_next_entry

```

```
266 0358 2 ;
267 0359 2
268 0360 2 $debug_print_lit ('entry');
269 0361 2
270 0362 2 $block_check (2, .volb, volb, 533);
271 0363 2 $logic_check (4, (exch$rtacp_verify_directory (.volb)), 183);
272 0364 2
273 0365 2
274 0366 2 seg = exch$rt11_dirseg_get (.volb, 1); ! Get the pointer to the dirty root segment
275 0367 2 $logic_check (2, (.seg NEQ 0), 151);
276 0368 2
277 0369 2 ! Figure out the maximum number of file entries which can fit into a single segment
278 0370 2
279 0371 2 max_ent_per_seg = (rt11$k_dirseglen - ! Length of entire segment
280 0372 2 rt11hdr$k_length - ! less the length of the segment header
281 0373 2 2) ! less two bytes for the end-of-segment marker
282 0374 2 / .ent_len; ! Divided by the length of a single entry
283 0375 2
284 0376 2 $trace_print_fao ('desired entries !UL, max_ent_per_seg !UL', .ent_cnt, .max_ent_per_seg);
285 0377 2
286 0378 2 ! Determine how many segments we need to leave two free entries per segment. This gives the final segment c
287 0379 2
288 0380 2 ent_per_seg = .max_ent_per_seg - 2; ! Leave room for two entries in each segment
289 0381 2 seg_cnt = (.ent_cnt / .ent_per_seg) + ! Need one extra segment if not even division
290 0382 2 (IF ((.ent_cnt MOD .ent_per_seg) NEQ 0) THEN 1 ELSE 0);
291 0383 2 IF .seg_cnt GTRU .seg [rt11hdr$w_num_segs]
292 0384 2 THEN
293 0385 2 seg_cnt = .seg [rt11hdr$w_num_segs]; ! We know that we can be at most one over the limit
294 0386 2
295 0387 2 ! Using the final segment count, now try to evenly distribute the entries around the segments. This gives t
296 0388 2 final entries per segment count.
297 0389 2
298 0390 2 ent_per_seg = (.ent_cnt / .seg_cnt) + ! Need one extra entry if not even division
299 0391 2 (IF ((.ent_cnt MOD .seg_cnt) NEQ 0) THEN 1 ELSE 0);
300 0392 2 $logic_check (4, (.ent_per_seg [EQU] .max_ent_per_seg), 180);
301 0393 2
302 0394 2 $trace_print_fao ('final segs !UL, goal ent_per_seg !UL', .seg_cnt, .ent_per_seg);
303 0395 2
304 0396 2 ! Grab a chunk of memory which will be used to build a clean copy of the directory. We haven't modified any
305 0397 2 yet, so there will be no problem with corrupting anything if this allocate fails (it will SIGNAL_STOP).
306 0398 2
307 0399 2 dir2 = exch$util_vm_allocate_zeroed (.seg_cnt * rt11$k_dirseglen);
```

```

309 0400 2 | OK, our new directory will contain .SEG_CNT segments each containing .ENT_PER_SEG entries, with possibly f
310 0401 2 | entries in the last segment. Start building a clean copy of the directory.
311 0402 2 |
312 0403 2 |
313 0404 2 | Set the segment header for the root segment. rt11hdr$w_high_seg won't be set until later
314 0405 2 |
315 0406 2 | seg2 = .dir2; | Get the pointer to the clean root segment
316 0407 2 | seg2 [rt11hdr$w_num_segs] = .seg [rt11hdr$w_num_segs]; | Haven't changed the total number of segs
317 0408 2 | seg2 [rt11hdr$w_next_seg] = (IF .seg_cnt EQ[ 1 THEN 0 ELSE 2); | We will link them in order
318 0409 2 | seg2 [rt11hdr$w_extra_bytes] = .seg [rt11hdr$w_extra_bytes]; | This stays the same
319 0410 2 | seg2 [rt11hdr$w_start_block] = .seg [rt11hdr$w_start_block]; | Same now, but may not be for other segment
320 0411 2 |
321 0412 2 | Set up the control variables for the transfer loop
322 0413 2 |
323 0414 2 | seg2_num = 1; | First segment in the clean directory
324 0415 2 | cur2 = .seg2 + rt11hdr$k_length; | Point at the first available entry position in the clean
325 0416 2 | ent2_cnt = 0; | No entries in the clean directory
326 0417 2 |
327 0418 2 | ctx = exch$util_rt11ctx_allocate (.volb, 0); | Get a context block for the search
328 0419 2 |
329 0420 2 | Loop through all the segments in the dirty directory, transferring entries to the clean directory
330 0421 2 |
331 0422 2 | flags = rtnxt$m_permanent OR rtnxt$m_empty OR rtnxt$m_tentative OR rtnxt$m_skip_check OR rtnxt$m_skip_expand
332 0423 2 | WHILE (cur = exch$rtacp_next_entry (.ctx, .flags)) NEQ 0
333 0424 2 | DO
334 0425 2 | BEGIN
335 0426 2 |
336 0427 2 | CASE .cur [rt11ent$v_type] FROM 0 TO rt11ent$m_typ_end_segment OF
337 0428 2 | SET
338 0429 2 |
339 0430 2 | [INRANGE, OUTRANGE] : | We had better give up before something happens
340 0431 2 | $logic_check (0, (false), 153);
341 0432 2 |
342 0433 2 | [rt11ent$m_typ_tentative, rt11ent$m_typ_empty, rt11ent$m_typ_permanent] :
343 0434 2 | BEGIN
344 0435 2 |
345 0436 2 | ! If we have filled this clean segment, then start a new segment before we do any moves
346 0437 2 |
347 0438 2 | IF (.ent2_cnt GEQU .ent_per_seg)
348 0439 2 | THEN
349 0440 2 | BEGIN
350 0441 2 |
351 0442 2 | ! Set the end-of-segment marker in the current clean segment
352 0443 2 |
353 0444 2 | $logic_check (3, (.cur2 LSSU .seg2 + rt11$k_dirseglen), 157); | We need room for marker
354 0445 2 | cur2 [rt11ent$b_type_byte] = rt11ent$m_typ_end_segment;
355 0446 2 |
356 0447 2 | ! Reset the pointers and counters for the clean segment
357 0448 2 |
358 0449 2 | ent2_cnt = 0; | No entries yet
359 0450 2 | seg2_num = .seg2_num + 1; | Adding one more segment
360 0451 2 | $logic_check (3, (.seg2_num LEQU .seg_cnt), 158); | which had better
361 0452 2 | seg2 = .seg2 + rt11$k_dirseglen; | Move to the new segment
362 0453 2 | cur2 = .seg2 + rt11hdr$k_length; | First entry in the seg
363 0454 2 |
364 0455 2 | ! Initialize the segment header for the new segment
365 0456 2 |

```

```

: 366 0457 5      seg2 [rt11hdr$w_num_segs] = .seg [rt11hdr$w_num_segs]; ! Same total # of segs
: 367 0458 6      seg2 [rt11hdr$w_next_seg] = (IF .seg2_num EQL .seg_cnt ! We will link them in order
: 368 0459 5      THEN 0 ELSE .seg2_num+1); ! marking the end with a 0
: 369 0460 5      seg2 [rt11hdr$w_extra_bytes] = .seg [rt11hdr$w_extra_bytes]; ! This stays the same
: 370 0461 5      seg2 [rt11hdr$w_start_block] = .ctx [rt11ctx$l_start_block]; ! First file starts here
: 371 0462 5
: 372 0463 4      END;
: 373 0464 4
: 374 0465 4      ! Transfer this entry to the clean segment
: 375 0466 4      !
: 376 0467 4      CH$MOVE (.ent_len, .cur, .cur2);
: 377 0468 4
: 378 0469 4      ! Bump the clean segment pointers
: 379 0470 4      !
: 380 0471 4      cur2 = .cur2 + .ent_len; ! Move the entry pointer
: 381 0472 4      ent2_cnt = .ent2_cnt + 1; ! Adjust the entry count for this segment
: 382 0473 4      ent_cnt = .ent_cnt - 1; ! Adjust the total entries remaining count, just a check
: 383 0474 4
: 384 0475 4      END;
: 385 0476 3      TES;
: 386 0477 2      END;
: 387 0478 2
: 388 0479 2      exch$util_rt11ctx_release (.ctx); ! Give the context block back
: 389 0480 2
: 390 0481 2      ! Set the end-of-segment marker in the final clean segment
: 391 0482 2      !
: 392 0483 2      $logic_check (3, (.cur2 LSSU .seg2 + rt11$k_dirseglen), 159); ! We need room for marker
: 393 0484 2      cur2 [rt11ent$b_type_byte] = rt11ent$m_typ_end_segment;
: 394 0485 2      $logic_check (3, ((.ent_cnt EQL 0)), 160); ! Make sure we moved everybody
: 395 0486 2
: 396 0487 2      ! Set the directory header info in the first and final segments
: 397 0488 2      !
: 398 0489 2      seg2 [rt11hdr$w_next_seg] = 0; ! Last segment is end of chain
: 399 0490 2      dir2 [rt11hdr$w_high_seg] = .seg2_num; ! Last segment is highest segment in use
: 400 0491 2
: 401 0492 2      !?? need to deal with the following situation (most likely to occur when /EXTRA=119):
: 402 0493 2      !?? attempting to completely fill each segment (e.g. seg_cnt=31 and ent_per_seg=4), however we end up
: 403 0494 2      !?? filling 23 segs with 4, the 24th seg gets 3, we quit but the insert fails because only one entry availa
: 404 0495 2      !?? in segment.
: 405 0496 2      !?? (If seg2_num < seg_cnt and < 2 free entries in last segment split the last segment into 2?
: 406 0497 2      !?? This doesn't help if free space is in an earlier segment unless you squeeze)

```

```

408 0498 2 ! Now that we have a clean "duplicate" of the directory, make it the real directory and update things on dis
409 0499 2 ! Since it is possible (in fact likely as the volume gets full) that some of the "clean" directory segments
410 0500 2 ! identical to "dirty" ones, we will compare each of the segments before moving and updating them.
411 0501 2
412 0502 2 seg2 = .dir2; ! Get the pointer to the clean root segment
413 0503 2
414 0504 2 INCRU num FROM 1 TO .seg2_num
415 0505 2 DO
416 0506 2 BEGIN
417 0507 2
418 0508 2 ! Compare the dirty segment with the clean one
419 0509 2
420 0510 2 IF CH$NEQ (rt11$k_dirseglen, .seg, rt11$k_dirseglen, .seg2)
421 0511 2 THEN
422 0512 2 BEGIN
423 0513 2
424 0514 2 CH$MOVE (rt11$k_dirseglen, .seg2, .seg); ! Move the clean segment on top of the dirty segment
425 0515 2 exch$rt11_dirseg_put (.volb, .num); ! Now write the segment to disk
426 0516 2
427 0517 2 END;
428 0518 2
429 0519 2 ! Move our segment pointers
430 0520 2
431 0521 2 seg = .seg + rt11$k_dirseglen;
432 0522 2 seg2 = .seg2 + rt11$k_dirseglen;
433 0523 2
434 0524 2 END;
435 0525 2
436 0526 2 ! Return the memory for the duplicate directory
437 0527 2
438 0528 2 exch$util_vm_release ((.seg_cnt * rt11$k_dirseglen), .dir2);
439 0529 2
440 0530 2 ! Check the new directory out
441 0531 2
442 0532 2 $logic_check (3, (exch$rtacp_verify_directory (.volb)), 184);
443 0533 2
444 0534 2 ! Now, consolidate again. This is necessary because we might have adjacent empty entries which can be colla
445 0535 2 ! We give the "2" flag to prevent another clean which would result in a recursive loop.
446 0536 2
447 0537 2 exch$rtacp_consolidate (.volb, 2);
448 0538 2
449 0539 2 RETURN;
450 0540 1 END;

```

		OFFC 00000		.ENTRY	EXCH\$RTACP CLEAN DIRECTORY, Save R2,R3,R4,-	: 0301
					R5,R6,R7,R8,R9,R10,R11	
SE		18	C2	SUBL2	#24, SP	
52	041B00F3	8F	DD	MOVL	#68878579, R2	: 0362
51	0215	8F	3C	MOVZWL	#533, R1	
50	04	AC	DD	MOVL	VOLB, R0	
	0000000G	EF	16	JSB	EXCH\$UTIL_BLOCK_CHECK	
		01	DD	PUSHL	#1	: 0366
	04	AC	DD	PUSHL	VOLB	

	00000000G	EF		02	FB	00020	CALLS	#2, EXCH\$RT11_DIRSEG_GET	
		5B		50	D0	00027	MOVL	RO, SEG	
				13	'2	0002A	BNEQ	1\$	0367
		7E	97	8F	9A	0002C	MOVZBL	#151, -(SP)	
				01	DD	00030	PUSHL	#1	
	00000000G		00000000G	8F	DD	00032	PUSHL	#EXCH\$BADLOGIC	
	50	000003F4		03	FB	00038	CALLS	#3, LIB\$STOP	0374
				AC	C7	0003F	DIVL3	ENT_LEN, #1012, MAX_ENT_PER_SEG	0380
				AO	9E	00048	MOVAB	-2(RO), ENT_PER_SEG	0381
7E				5A	C7	0004C	DIVL3	ENT_PER_SEG, ENT_CNT, R1	0382
50				01	7A	00051	EMUL	#1, ENT_CNT, #0, -(SP)	
				5A	7B	00057	EDIV	ENT_PER_SEG, (SP)+, RO, RO	
				50	D5	0005C	TSTL	RO	
				05	13	0005E	BEQL	2\$	
				01	D0	00060	MOVL	#1, RO	
				02	11	00063	BRB	3\$	
				50	D4	00065	CLRL	RO	
58				50	C1	00067	ADDL3	RO, R1, SEG_CNT	0383
				00	ED	0006B	CMPZV	#0, #16, (SEG), SEG_CNT	
				03	1E	00070	BGEQU	4\$	
				6B	3C	00072	MOVZWL	(SEG), SEG_CNT	0385
				58	C7	00075	DIVL3	SEG_CNT, ENT_CNT, R1	0390
7E				01	7A	0007A	EMUL	#1, ENT_CNT, #0, -(SP)	0391
50				58	7B	00080	EDIV	SEG_CNT, (SP)+, RO, RO	
				50	D5	00085	TSTL	RO	
				05	13	00087	BEQL	5\$	
				01	D0	00089	MOVL	#1, RO	
				02	11	0008C	BRB	6\$	
				50	D4	0008E	CLRL	RO	
				50	C1	00090	ADDL3	RO, R1, ENT_PER_SEG	
14	SA			0A	78	00094	ASHL	#10, SEG_CNT, 20(SP)	0399
	AE			AE	DD	00099	PUSHL	20(SP)	
				01	FB	0009C	CALLS	#1, EXCH\$UTIL_VM_ALLOCATE_ZEROED	
	00000000G	EF		50	D0	000A3	MOVL	RO, DIR2	
		OC		AE	D0	000A7	MOVL	DIR2, SEG2	0406
				6B	B0	000AB	MOVW	(SEG), (SEG2)	0407
				58	D1	000AE	C MPL	SEG_CNT, #1	0408
				04	12	000B1	BNEQ	7\$	
				50	D4	000B3	CLRL	RO	
				03	11	000B5	BRB	8\$	
				02	D0	000B7	MOVL	#2, RO	
				50	B0	000BA	MOVW	RO, 2(SEG2)	
				06	AB	000BE	MOVL	6(SEG), 6(SEG2)	0409
				01	D0	000C3	MOVL	#1, SEG2_NUM	0414
				0A	A6	000C6	MOVAB	10(R6), CUR2	0415
				08	AE	D4	000CA	CLRL	ENT2 CNT
				7E	D4	000CD	CLRL	-(SP)	0416
				04	AC	DD	000CF	PUSHL	VOLB
	00000000G	EF		02	FB	000D2	CALLS	#2, EXCH\$UTIL_RT11CTX_ALLOCATE	
		6E		50	D0	000D9	MOVL	RO, CTX	
				37	D0	000DC	MOVL	#55, FLAGS	0422
				AE	DD	000E0	PUSHL	FLAGS	0423
				04	AE	DD	000E3	PUSHL	CTX
				02	FB	000E6	CALLS	#2, EXCH\$RTACP_NEXT_ENTRY	
				50	D0	000EB	MOVL	RO, CUR	
				03	12	000EF	BNEQ	10\$	
				0084	31	000F1	BRW	18\$	

7E	7E	04	AE	01	C1	000F4	10\$:	ADDL3	#1, CUR, -(SP)	0427		
	9E		04	00	EF	000F9		EXTZV	#0, #4, @ (SP)+, -(SP)			
	08		00	8E	CF	000FE		CASEL	(SP)+, #0, #8			
0012	0027		0027	0012		00102	11\$:	.WORD	12\$-11\$,-			
0012	0012		0012	0027		0010A			14\$-11\$,-			
				0012		00112			14\$-11\$,-			
									12\$-11\$,-			
									14\$-11\$,-			
									12\$-11\$,-			
									12\$-11\$,-			
									12\$-11\$,-			
									12\$-11\$			
		7E		99	8F	9A	00114	12\$:	MOVZBL	#153, -(SP)	0431	
					01	DD	00118		PUSHL	#1		
					8F	DD	0011A		PUSHL	#EXCH\$ BADLOGIC		
	00000000G	00			03	FB	00120		CALLS	#3, LIB\$STOP		
					B7	11	00127	13\$:	BRB	9\$		
		5A		08	AE	D1	00129	14\$:	CMP	ENT2_CNT, ENT_PER_SEG	0438	
					37	1F	0012D		BLSSU	17\$		
		01	A7		08	90	0012F		MOV	#8, 1(CUR2)	0445	
					08	AE	00133		CLRL	ENT2_CNT	0449	
					59	D6	00136		INCL	SEG2_NUM	0450	
		56		0400	C6	9E	00138		MOV	1024(R6), SEG2	0452	
		57		0A	A6	9E	0013D		MOV	10(R6), CUR2	0453	
		66			6B	B0	00141		MOV	(SEG), (SEG2)	0457	
		58			59	D1	00144		CMP	SEG2_NUM, SEG_CNT	0458	
					04	12	00147		BNEQ	15\$		
					50	D4	00149		CLRL	R0		
					04	11	0014B		BRB	16\$		
		50		01	A9	9E	0014D	15\$:	MOV	1(R9), R0	0459	
		02	A6		50	B0	00151	16\$:	MOV	R0, 2(SEG2)	0458	
		06	A6	06	AB	B0	00155		MOV	6(SEG), 6(SEG2)	0460	
50		6E	00000072		8F	C1	0015A		ADDL3	#114, CTX, R0	0461	
		08	A6		60	B0	00162		MOV	(R0), 8(SEG2)		
67		04	BE		0C	AC	28	00166	17\$:	MOV	ENT_LEN, @CUR, (CUR2)	0467
			57		0C	AC	C0	0016C		ADDL2	ENT_LEN, CUR2	0471
					08	AE	D6	00170		INCL	ENT2_CNT	0472
					08	AC	D7	00173		DECL	ENT_CNT	0473
					AF	11	00176		BRB	13\$	0423	
					6E	DD	00178	18\$:	PUSHL	CTX	0479	
	00000000G		EF		01	FB	0017A		CALLS	#1, EXCH\$UTIL_RT11CTX_RELEASE		
		01	A7		08	90	00181		MOV	#8, 1(CUR2)	0484	
					A6	B4	00185		CLRW	2(SEG2)	0489	
50		0C	AE	02	04	C1	00188		ADDL3	#4, DIR2, R0	0490	
			60		59	B0	0018D		MOV	SEG2_NUM, (R0)		
			56	0C	AE	D0	00190		MOV	DIR2, SEG2	0502	
			58		01	D0	00194		MOV	#1, NUM	0504	
					26	11	00197		BRB	21\$		
66		6B	0400		8F	29	00199	19\$:	CMP	#1024, (SEG), (SEG2)	0510	
					12	13	0019F		BEQ	20\$		
6B		66	0400		8F	28	001A1		MOV	#1024, (SEG2), (SEG)	0514	
					58	DD	001A7		PUSHL	NUM	0515	
					04	AC	DD	001A9		PUSHL	VOLB	
	00000000G		EF		02	FB	001AC		CALLS	#2, EXCH\$RT11_DIRSEG_PUT		
			5B	0400	CB	9E	001B3	20\$:	MOV	1024(R11), SEG	0511	
			56	0400	C6	9E	001B8		MOV	1024(R6), SEG2		
					58	D6	001BD		INCL	NUM		

EXCH\$RTACP  
V04-000

RT11 directory routines  
exch\$rtacp\_clean\_directory (volb, ent\_cnt, ent\_

L 4  
16-Sep-1984 01:19:05  
14-Sep-1984 12:29:08

VAX-11 Bliss-32 V4.0-742  
[EXCHNG.SRC]EXCRTACP.B32;1

Page 14  
(7)

EXI  
VOI

59		58	D1	001BF	21\$:	CMPL	NUM, SEG2_NUM	:	
		D5	1B	001C2		BLEQU	19\$	:	
		OC	AE	DD	001C4	PUSHL	DIR2	:	052B
		18	AE	DD	001C7	PUSHL	24(SP)	:	
00000000G	EF		02	FB	001CA	CALLS	#2, EXCH\$UTIL_VM_RELEASE	:	
			02	DD	001D1	PUSHL	#2	:	0537
		04	AC	DD	001D3	PUSHL	VOLB	:	
0000V	CF		02	FB	001D6	CALLS	#2, EXCH\$RTACP_CONSOLIDATE	:	
			04	001DB		RET		:	0540

; Routine Size: 476 bytes, Routine Base: EXCH\$RTACP\_CODE + 00B2



```
452 0541 1 GLOBAL ROUTINE exch$rtacp_consolidate (volb : $ref_bblock, clean) = %SBTTL 'exch$rtacp_consolidate (volb
453 0542 2 BEGIN
454 0543 2 ++
455 0544 2
456 0545 2 FUNCTIONAL DESCRIPTION:
457 0546 2
458 0547 2 Make a pass to compress the directory. The following types of entries are unnecessary:
459 0548 2
460 0549 2 Multiple consecutive empty entries
461 0550 2 Tentative entries
462 0551 2 Empties of length 0 following a permanent entry
463 0552 2
464 0553 2 INPUTS:
465 0554 2
466 0555 2 volb - pointer to volb which has been connected to the RT-11 device
467 0556 2 clean - flag telling whether to restructure the directory, with the following meanings:
468 0557 2 0 - don't force a restructure, but allow it if find segment with single entry
469 0558 2 1 - force a restructure regardless
470 0559 2 2 - don't allow a restructure
471 0560 2
472 0561 2 IMPLICIT INPUTS:
473 0562 2
474 0563 2 none
475 0564 2
476 0565 2 OUTPUTS:
477 0566 2
478 0567 2 none
479 0568 2
480 0569 2 IMPLICIT OUTPUTS:
481 0570 2
482 0571 2 none
483 0572 2
484 0573 2 ROUTINE VALUE:
485 0574 2
486 0575 2 true if succeeded, false or error status if failed
487 0576 2
488 0577 2 SIDE EFFECTS:
489 0578 2
490 0579 2 error conditions will be signaled
491 0580 2 --
492 0581 2
493 0582 2 $dbgtrc_prefix ('exch$rtacp_consolidate> ');
```

```

: 495 0583 2 LOCAL
: 496 0584 2     seg : $ref_bblock,      ! a pointer to the current directory segment
: 497 0585 2     cur : $ref_bblock,    ! a pointer to the current directory entry
: 498 0586 2     prv : $ref_bblock,    ! a pointer to the previous directory entry
: 499 0587 2     segment_modified : BITVECTOR [32], ! bits to say whether we need to write out directory segment
: 500 0588 2     seg_num,              ! current segment number
: 501 0589 2     end_segment_seen,     ! we have seen an end of segment marker
: 502 0590 2     segment_with_single_entry_seen, ! a segment was consolidated to a single entry
: 503 0591 2     ent_len,              ! length of a single directory entry
: 504 0592 2     ent_cnt,              ! number of entries in the segment
: 505 0593 2     ;
: 506 0594 2
: 507 0595 2 BIND
: 508 0596 2     modified_segments = segment_modified ! map a longword onto the bitvector
: 509 0597 2     ;
: 510 0598 2
: 511 0599 2 $trace_print_fao ('entry');
: 512 0600 2
: 513 0601 2 $block_check (2, .volb, volb, 551);
: 514 0602 2 $logic_check (4, (exch$rtacp_verify_directory (.volb)), 211);
: 515 0603 2
: 516 0604 2 ' Set up some initial conditions
: 517 0605 2 .
: 518 0606 2 modified_segments = 0;      ! No directory segments have been modified, clear all bits
: 519 0607 2 ent_cnt = 0;              ! No entries yet
: 520 0608 2 segment_with_single_entry_seen = false; ! No segment was consolidated to a single entry

```

```

: 522 0609 2 | Scan the directory, compressing unnecessary file entries
: 523 0610 2 |
: 524 0611 2 | Start with the first directory segment
: 525 0612 2 |
: 526 0613 2 seg_num = 1;
: 527 0614 2 |
: 528 0615 2 | Loop through all the segments in the directory
: 529 0616 2 |
: 530 0617 2 WHILE .seg_num NEQ 0
: 531 0618 2 DO
: 532 0619 2 BEGIN
: 533 0620 2 |
: 534 0621 2 | Get a pointer to the current segment
: 535 0622 2 |
: 536 0623 2 seg = exch$rt11_dirseg_get (.volb, .seg_num);
: 537 0624 2 $logic_check (2, (.seg NEQ 0), 152);
: 538 0625 2 ent_len = rt11ent$k_length + .seg [rt11hdr$w_extra_bytes];
: 539 0626 2 |
: 540 0627 2 | Get a pointer to the first directory entry, delete it if it is tentative
: 541 0628 2 |
: 542 0629 2 prv = .seg + rt11hdr$k_length;
: 543 0630 2 IF (.prv [rt11ent$v_type] EQL rt11ent$m_typ_tentative)
: 544 0631 2 THEN
: 545 0632 2 BEGIN
: 546 0633 2 segment_modified [.seg_num] = true;
: 547 0634 2 prv [rt11ent$b_type_byte] = rt11ent$m_typ_empty; ! Zap whole byte to clear protect bit
: 548 0635 2 END;
: 549 0636 2 |
: 550 0637 2 | We've taken care of the first entry, now starting with the second scan the rest of this segment. If e
: 551 0638 2 | of the first two entries is the end marker we don't need to scan
: 552 0639 2 |
: 553 0640 2 cur = .prv + .ent_len; ! Add the length of one entry
: 554 0641 2 IF (.prv [rt11ent$v_type] NEQ rt11ent$m_typ_end_segment)
: 555 0642 2 AND
: 556 0643 2 (.cur [rt11ent$v_type] NEQ rt11ent$m_typ_end_segment)
: 557 0644 2 THEN
: 558 0645 2 BEGIN
: 559 0646 2 end_segment_seen = false;
: 560 0647 2 WHILE (.cur LSSU (.seg + rt11$k_dirseglen))
: 561 0648 2 DO
: 562 0649 2 BEGIN
: 563 0650 2 |
: 564 0651 2 | If tentative, turn it into an empty before we move forward
: 565 0652 2 |
: 566 0653 2 IF (.cur [rt11ent$v_type] EQL rt11ent$m_typ_tentative)
: 567 0654 2 THEN
: 568 0655 2 BEGIN
: 569 0656 2 segment_modified [.seg_num] = true;
: 570 0657 2 cur [rt11ent$b_type_byte] = rt11ent$m_typ_empty; ! Zap whole byte to clear protect bi
: 571 0658 2 END;
: 572 0659 2 |
: 573 0660 2 | Advance our count
: 574 0661 2 |
: 575 0662 2 CASE .cur [rt11ent$v_type] FROM 0 TO rt11ent$m_typ_end_segment OF
: 576 0663 2 SET
: 577 0664 2 [rt11ent$m_typ_tentative] :
: 578 0665 2 $logic_check (0, (false), 148);
```

```
.. 579 0666 5
580 0667 5 [rt1lent$m_typ_permanent] :
581 0668 5 ;
582 0669 5 ;
583 0670 5 [rt1lent$m_typ_empty] :
584 0671 6 BEGIN
585 0672 6
586 0673 6 ! If the previous entry is permanent and the size of this empty is zero we can get rid o
587 0674 6 entry.
588 0675 6
589 0676 7 IF (.prv [rt1lent$v_type] EQL rt1lent$m_typ_permanent)
590 0677 6 AND
591 0678 7 (.cur [rt1lent$w_blocks] EQL 0)
592 0679 6 THEN
593 0680 7 BEGIN
594 0681 7 LOCAL
595 0682 7 sl; ! Length of directory to slide down one entry
596 0683 7
597 0684 7 ! Slide the rest of the segment down one entry
598 0685 7
599 0686 7 sl = rt11$k_dirseglen - .ent_len - (.cur - .seg); ! Length of segment remainin
600 0687 7 CH$MOVE (.s[, .cur + .ent_len, .cur);
601 0688 7 segment_modified [.seg_num] = true;
602 0689 7 cur = .cur - .ent_len; ! Back up so that next pass sees the slid down entry
603 0690 7 END
604 0691 7
605 0692 7 ! Or if the previous entry is also empty we can combine these two into a single empty en
606 0693 7
607 0694 7 ELSE IF (.prv [rt1lent$v_type] EQL rt1lent$m_typ_empty)
608 0695 6 THEN
609 0696 7 BEGIN
610 0697 7 LOCAL
611 0698 7 sl; ! Length of directory to slide down one entry
612 0699 7
613 0700 7 ! Add the size of this empty to the previous empty, plus count it in the total
614 0701 7
615 0702 7 prv [rt1lent$w_blocks] = .prv [rt1lent$w_blocks] + .cur [rt1lent$w_blocks];
616 0703 7
617 0704 7 ! Slide the rest of the segment down one entry
618 0705 7
619 0706 7 sl = rt11$k_dirseglen - .ent_len - (.cur - .seg); ! Length of segment remainin
620 0707 7 CH$MOVE (.s[, .cur + .ent_len, .cur);
621 0708 7 segment_modified [.seg_num] = true; ! Remember that we have chan
622 0709 7 cur = .cur - .ent_len; ! Back up so that next pass sees the slid down entry
623 0710 6 END;
624 0711 5 END;
625 0712 5
626 0713 5 [rt1lent$m_typ_end_segment] :
627 0714 6 BEGIN
628 0715 6 end_segment_seen = true;
629 0716 6 EXITLOOP;
630 0717 5 END;
631 0718 5
632 0719 5 ! Volumes with corrupted directories should have been write-locked when the directory was verifi
633 0720 5 during mount. Therefore we must assume an I/O error or programming error within EXCHANGE has
634 0721 5 corrupted this directory.
635 0722 5
```

```

: 636      0723 5      [INRANGE, OUNTRANGE] :      ! We had better give up before something hap
: 637      0724 6      BEGIN
: 638      0725 6      exch$rt11_dirseg_flush (.volb, .modified_segments); ! Write out any changes up til now
: 639      0726 6      $exch_signal_stop (exch$_rt11_baddirect, 2, .volb [volb$l_vol_ident_len], volb [volb$t_v
: 640      0727 5      END;
: 641      0728 5
: 642      0729 5      TES;
: 643      0730 5
: 644      0731 5      ! Make the current the previous entry and skip to the next
: 645      0732 5      !
: 646      0733 5      prv = .cur;
: 647      0734 5      cur = .cur + .ent_len;
: 648      0735 5
: 649      0736 4      END;
: 650      0737 4
: 651      0738 4      ! Volumes with corrupted directories should have been write-locked when the directory was verified
: 652      0739 4      during mount. Therefore we must assume an I/O error or programming error within EXCHANGE has
: 653      0740 4      corrupted this directory.
: 654      0741 4
: 655      0742 5      IF NOT (.end_segment_seen)
: 656      0743 4      THEN
: 657      0744 5      BEGIN
: 658      0745 5      exch$rt11_dirseg_flush (.volb, .modified_segments); ! Write out any changes up til now
: 659      0746 5      $exch_signal_stop (exch$_rt11_baddirect, 2, .volb [volb$l_vol_ident_len], volb [volb$t_vol_ident
: 660      0747 4      END;
: 661      0748 4      END;
: 662      0749 4
: 663      0750 3      ! Count the number of entries. CUR points to the end marker.
: 664      0751 3
: 665      0752 5      (BEGIN LOCAL cnt_this_seg;
: 666      0753 5      cnt_this_seg = (.cur - (.seg + rt11hdr$k_length)) / .ent_len;      ! Divide space used by length of ent
: 667      0754 5      ent_cnt = .ent_cnt + .cnt_this_seg;      ! Divide space by length
: 668      0755 5      IF .cnt_this_seg EQL 1      ! This segment has a single entry
: 669      0756 5      THEN
: 670      0757 5      segment_with_single_entry_seen = true;
: 671      0758 5      END);
: 672      0759 3      $logic_check (4, (((.cur - (.seg + rt11hdr$k_length)) MOD .ent_len) EQL 0), 150);      ! Better be even
: 673      0760 3
: 674      0761 3      ! Skip to the next segment
: 675      0762 3      !
: 676      0763 3      seg_num = .seg [rt11hdr$w_next_seg];
: 677      0764 3
: 678      0765 2      END;
: 679      0766 2
: 680      0767 2      ! Now that we are done, write out any of the segments that we have modified
: 681      0768 2      !
: 682      0769 2      exch$rt11_dirseg_flush (.volb, .modified_segments);
: 683      0770 2
: 684      0771 2      $logic_check (3, (exch$rtacp_verify_directory (.volb)), 191);
: 685      0772 2
: 686      0773 2      ! Now clean it up if one of the flags is set
: 687      0774 2      !
: 688      0775 2      IF (.clean EQL 1)
: 689      0776 2      OR
: 690      0777 2      (.segment_with_single_entry_seen
: 691      0778 2      AND
: 692      0779 3      .ent_cnt GTRU 1      ! Avoid recursive loops when down to one entry

```

```

: 693 0780 3   AND
: 694 0781 3   .clean NEQ 2)
: 695 0782 2   THEN
: 696 0783 2   exch$rtacp_clean_directory (.volb, .ent_cnt, .ent_len);
: 697 0784 2
: 698 0785 2   RETURN true;
: 699 0786 1   END;
! Even then, don't clean if it has been prohibited

```

				OFFC	00000			.EXTRN	EXCH\$ RT11_BADDIRECT	
								.EXTRN	LIB\$STOP	
								.ENTRY	EXCH\$RTACP CONSOLIDATE, Save R2,R3,R4,R5,-	0541
			5E		18	C2	00002	SUBL2	#24, SP	
			59	04	AC	DO	00005	MOVL	VOLB, R9	0601
			52	041B00F3	8F	DO	00009	MOVL	#68878579, R2	
			51	0227	8F	3C	00010	MOVZWL	#551, R1	
			50		59	DO	00015	MOVL	R9, R0	
				00000000G	EF	16	00018	JSB	EXCH\$UTIL_BLOCK_CHECK	
					08	AE	7C	CLRQ	ENT CNT	0607
					04	AE	D4	CLRL	SEGMENT_WITH_SINGLE_ENTRY_SEEN	0608
			5B		01	DO	00024	MOVL	#1, SEG_NUM	0613
					03	12	00027	BNEQ	2\$	0617
					0132	31	00029	BRW	24\$	
				0A00	8F	BB	0002C	PUSHR	#^M<R9,R11>	0623
				00000000G	02	FB	00030	CALLS	#2, EXCH\$RT11_DIRSEG_GET	
			5A		50	DO	00037	MOVL	R0, SEG	
					13	12	0003A	BNEQ	3\$	0624
			7E	98	8F	9A	0003C	MOVZBL	#152, -(SP)	
					01	DD	00040	PUSHL	#1	
				00000000G	8F	DD	00042	PUSHL	#EXCH\$ BADLOGIC	
			00		03	FB	00048	CALLS	#3, LIB\$STOP	
			58	06	AA	3C	0004F	MOVZWL	6(SEG), ENT_LEN	0625
			58		0E	C0	00053	ADDL2	#14, ENT_LEN	
			14		AE	0A	9E	MOVAB	10(R10), -20(SP)	0629
			57	14	AE	DO	0005B	MOVL	20(SP), PRV	
01	01	A7	04		00	ED	0005F	CMPZV	#0, #4, 1(PRV), #1	0630
					09	12	00065	BNEQ	5\$	
					5B	E2	00067	BBSS	SEG_NUM, SEGMENT_MODIFIED, 4\$	0633
					02	90	0006C	MOVAB	#2, 1(PRV)	0634
					58	C1	00070	ADDL3	ENT_LEN, PRV, CUR	0640
08	01	A7	04		00	ED	00074	CMPZV	#0, #4, 1(PRV), #8	0641
					06	13	0007A	BEQL	6\$	
08	01	A6	04		00	ED	0007C	CMPZV	#0, #4, 1(CUR), #8	0643
					03	12	00082	BNEQ	7\$	
					00BB	31	00084	BRW	22\$	
					6E	D4	00087	CLRL	END SEGMENT SEEN	0646
			10	AE	CA	9E	00089	MOVAB	1024(R10), T6(SP)	0647
			10	AE	56	D1	0008F	CML	CUR, 16(SP)	
					7D	1E	00093	BGEQU	18\$	
01	01	A6	04		00	ED	00095	CMPZV	#0, #4, 1(CUR), #1	0653
					09	12	0009B	BNEQ	10\$	
					5B	E2	0009D	BBSS	SEG_NUM, SEGMENT_MODIFIED, 9\$	0656
					02	90	000A2	MOVAB	#2, 1(CUR)	0657
7E	01	A6	04		00	EF	000A6	EXTZV	#0, #4, 1(CUR), -(SP)	0662

0070 0070	08 0029 0070	00 0014 0070	8E 0070 0064 005F	CF	000AC 000B0 000B8 000C0	11\$:	CASEL .WORD	(SP)+ #0, #8 21\$-11\$,- 12\$-11\$,- 13\$-11\$,- 21\$-11\$,- 19\$-11\$,- 21\$-11\$,- 21\$-11\$,- 21\$-11\$,- 17\$-11\$		
					5C 11 000C2		BRB	21\$		0725
		7E	94		8F 9A 000C4	12\$:	MOVZBL	#148, -(SP)		0665
					01 DD 000C8		PUSHL	#1		
					8F DD 000CA		PUSHL	#EXCH\$ BADLOGIC		
		00000000G	00		03 FB 000D0		CALLS	#3, LIB\$STOP		
					3B 11 000D7		BRB	19\$		
04	01	A7	04		00 ED 000D9	13\$:	CMPZV	#0, #4, 1(PRV), #4		0676
					05 12 000DF		BNEQ	14\$		
					08 A6 B5 000E1		TSTW	8(CUR)		0678
					0D 13 000E4		BEQL	15\$		
02	01	A7	04		00 ED 000E6	14\$:	CMPZV	#0, #4, 1(PRV), #2		0694
					26 12 000EC		BNEQ	19\$		
		08	A7	08	A6 A0 000EE		ADDW2	8(CUR), 8(PRV)		0702
		50	56		5A C3 000F3	15\$:	SUBL3	SEG, CUR, RO		0706
			50		FC00 C0 8 9E 000F7		MOVAB	-1024(RO)[ENT_LEN], SL		
			50		50 CE 000FD		MNEGL	SL, SL		
		66	6846		50 28 00100		MOV3	SL, (ENT_LEN)[CUR], (CUR)		0707
		00	OC		5B E2 00105		BBSS	SEG_NUM, -SEGMENT_MODIFIED, 16\$		0708
			56		58 C2 0010A	16\$:	SUBL2	ENT_LEN, CUR		0709
					05 11 0010D		BRB	19\$		0662
		6E			01 D0 0010F	17\$:	MOVL	#1, END_SEGMENT_SEEN		0715
					09 11 00112	18\$:	BRB	20\$		0714
		57			56 D0 00114	19\$:	MOVL	CUR, PRV		0733
		56			58 C0 00117		ADDL2	ENT_LEN, CUR		0734
					FF72 31 0011A		BRW	8\$		0647
		22			6E E8 0011D	20\$:	BLBS	END_SEGMENT_SEEN, 22\$		0742
					OC AE DD 00120	21\$:	PUSHL	MODIFIED_SEGMENTS		0745
					59 DD 00123		PUSHL	R9		
		00000000G	EF		02 FB 00125		CALLS	#2, EXCH\$RT11_DIRSEG_FLUSH		
					69 A9 9F 0012C		PUSHAB	105(R9)		0746
					65 A9 DD 0012F		PUSHL	101(R9)		
					02 DD 00132		PUSHL	#2		
					8F DD 00134		PUSHL	#EXCH\$ RT11 BADDIRECT		
		00000000G	00		04 FB 0013A		CALLS	#4, LIB\$STOP		
					04 00141		RET			
		50	56	14	AE C3 00142	22\$:	SUBL3	20(SP), CUR, RO		0753
			50		58 C6 00147		DIVL2	ENT_LEN, CNT_THIS_SEG		
		08	AE		50 C0 0014A		ADDL2	CNT_THIS_SEG, ENT_CNT		0754
			01		50 D1 0014E		CML	CNT_THIS_SEG, #1		0755
					04 12 00151		BNEQ	23\$		
		04	AE		01 D0 00153		MOVL	#1, SEGMENT_WITH_SINGLE_ENTRY_SEEN		0757
			5B		02 AA 3C 00157	23\$:	MOVZWL	2(SEG), SEG_NUM		0763
					FEC9 31 0015B		BRW	1\$		0617
					OC AE DD 0015E	24\$:	PUSHL	MODIFIED_SEGMENTS		0769
					59 DD 00161		PUSHL	R9		
		00000000G	EF		02 FB 00163		CALLS	#2, EXCH\$RT11_DIRSEG_FLUSH		
			01		08 AC D1 0016A		CML	CLEAN, #1		0775

EXCH\$RTACP  
V04-000

RT11 directory routines  
exch\$rtacp\_consolidate (volb, clean)

G 5  
16-Sep-1984 01:19:05  
14-Sep-1984 12:29:08

VAX-11 Bliss-32 V4.0-742  
[EXCHNG.SRC]EXCRTACP.832;1

Page 22  
(10)

		10	13	0016E	BEQL	25\$		
	18	04	AE	E9 00170	BLBC	SEGMENT_WITH_SINGLE_ENTRY_SEEN, 26\$	:	0777
	01	08	AE	D1 00174	CMPL	ENT_CNT, #1	:	0779
		12	1B	00178	BLEQU	26\$	:	
	02	08	AC	D1 0017A	CMPL	CLEAN, #2	:	0781
		0C	13	0017E	BEQL	26\$	:	
		58	DD	00180 25\$:	PUSHL	ENT_LEN	:	0783
		0C	AE	DD 00182	PUSHL	ENT_CNT	:	
		59	DD	00185	PUSHL	R9	:	
FC98	CF	03	FB	00187	CALLS	#3, EXCH\$RTACP_CLEAN_DIRECTORY	:	
	50	01	DD	0018C 26\$:	MOVL	#1, R0	:	0785
		04	0018F		RET		:	0786

: Routine Size: 400 bytes, Routine Base: EXCH\$RTACP\_CODE + 028E



```

0787 1 GLOBAL ROUTINE exch$rtacp_find_empty_area (ctx : $ref_bblock, %SBTTL 'exch$rtacp_find_empty_area (ctx, blo
0788 1                                     blocks, start) =
0789 2 BEGIN
0790 2 ++
0791 2
0792 2 FUNCTIONAL DESCRIPTION:
0793 2
0794 2     First, have EXCH$RT11_CONSOLIDATE make a pass to compress the directory.
0795 2
0796 2     Second, make a pass to prepare to enter a new file. The BLOCKS parameter gives the size of the area
0797 2     is needed. We will return the address of the first empty area which is as close to this size as we
0798 2     get. We will also put a zero-length empty area after this area so that the rt-11 close routine can
0799 2     move any excess blocks into the second entry.
0800 2
0801 2 INPUTS:
0802 2
0803 2     ctx      - pointer to rt11 context block
0804 2     blocks  - number of blocks needed, 0 means largest possible entry
0805 2     start   - pbn of block to start allocation (blocks parameter ignored if start <> 0)
0806 2     (dummy) - this routine will call itself a second time if it is unable to satisfy the request. Recur
0807 2               calls are flagged with a 4th dummy parameter, ACTUALCOUNT () is examined to prevent loops
0808 2
0809 2 IMPLICIT INPUTS:
0810 2
0811 2     none
0812 2
0813 2 OUTPUTS:
0814 2
0815 2     none
0816 2
0817 2 IMPLICIT OUTPUTS:
0818 2
0819 2     none
0820 2
0821 2 ROUTINE VALUE:
0822 2
0823 2     true if able to find a suitable empty area, false or error status if failed
0824 2
0825 2 SIDE EFFECTS:
0826 2
0827 2     error conditions will be signaled
0828 2
0829 2 --
0830 2 $dbgtrc_prefix ('exch$rtacp_find_empty_area> ');
0831 2
0832 2 LOCAL
0833 2     seg : $ref_bblock,      ! a pointer to the current directory segment
0834 2     cur : $ref_bblock,      ! a pointer to the current directory entry
0835 2     exact_match,           ! found an entry of exactly the correct length
0836 2     ent_len,               ! length of a single directory entry
0837 2     flags,                 ! a temporary for the flags mask
0838 2     mat_ent : $ref_bblock,  ! address of current match
0839 2     mat_blk,               ! start pbn of current match
0840 2     mat_len,               ! size of current match
0841 2     mat_seg,               ! segment number containing the match
0842 2     recursive,            ! flag that we have been called from here
0843 2     status

```

```
758 0844 2 ;
759 0845 2 ;
760 0846 2 BIND
761 0847 2     volb = ctx [rt11ctx$a_assoc_volb] : $ref_bblock
762 0848 2 ;
763 0849 2 ;
764 0850 2 BUILTIN
765 0851 2     ACTUALCOUNT
766 0852 2 ;
767 0853 2 ;
768 0854 2 ;
769 0855 2 $trace_print_fao ('entry - ctx !XL, blocks !UL, start !UL', .ctx, .blocks, .start);
770 0856 2 ;
771 0857 2 $block_check (2, .ctx, rt11ctx, 535);
772 0858 2 $block_check (2, .volb, volb, 530);
773 0859 2 ;
774 0860 2 ! Set up some initial conditions
775 0861 2 ;
776 0862 2 ctx [rt11ctx$a_ent_address] = 0;           ! Set the output to an invalid address
777 0863 2 ctx [rt11ctx$l_seg_number] = 0;         ! Reset to scan from the start of the directory
778 0864 2 mat_len = 0;                             ! No length of any matched entry
779 0865 2 recursive = (IF ACTUALCOUNT () EQL 4 THEN 1 ELSE 0);
780 0866 2 ;
781 0867 2 ! Compress the directory into a known, clear state. Consolidate will do any verification we need. The seco
782 0868 2 ! parameter determines whether a directory restructuring should take place.
783 0869 2 ;
784 0870 2 IF NOT (status = exch$rtacp_consolidate (.volb, .recursive))
785 0871 2 THEN
786 0872 2     RETURN .status;
787 0873 2 ;
788 0874 2 ! Loop through all the segments in the directory
789 0875 2 ;
790 0876 2 exact_match = false;                         ! We haven't seen an empty entry of exactly the correct size
791 0877 2 flags = rtnxt$m_empty OR rtnxt$m_skip_check OR rtnxt$m_skip_expand;
792 0878 2 WHILE ((cur = exch$rtacp_next_entry (.ctx, .flags)) NEQ 0)
793 0879 2 DO
794 0880 2     BEGIN
795 0881 2     LOCAL
796 0882 2     len;
797 0883 2 ;
798 0884 2     len = .cur [rt11ent$w_blocks];           ! Put the length into a local for speed
799 0885 2 ;
800 0886 2     ! If we are looking for a particular start block, see if this empty area contains the start
801 0887 2     ;
802 0888 2     IF .start NEQ 0
803 0889 2     THEN
804 0890 2     BEGIN
805 0891 2     IF .start LSSU .ctx [rt11ctx$l_start_block]
806 0892 2     THEN
807 0893 2     RETURN exch$stnotavail                 ! Already past the block, we will never find it
808 0894 2     ELSE
809 0895 2     BEGIN
810 0896 2     IF .start LEQU (.len + .ctx [rt11ctx$l_start_block]) ! Requested block inside this area
811 0897 2     THEN
812 0898 2     BEGIN
813 0899 2     exact_match = true;                   ! Set the flag to stop scanning
814 0900 2     END

```

```
815 0901 6      | Save the entry as a match. Note that this and the following three saves are written
816 0902 6      | so that the optimizer can collapse them into a single segment of object code.
817 0903 6
818 0904 6      mat_len = .len;      | Save the length of this entry,
819 0905 6      mat_ent = .cur;      | its address,
820 0906 6      mat_seg = .ctx [rt11ctx$l_seg_number]; | its segment number,
821 0907 6      mat_blk = .ctx [rt11ctx$l_start_block]; | and the pbn where the free space starts
822 0908 5      END;
823 0909 4      END;
824 0910 4      END
825 0911 4
826 0912 4      | We are looking for any old empty area, see if the size is appropriate
827 0913 4
828 0914 3      ELSE IF .len GEQU .blocks      | If this length is a candidate, then examine it more closely
829 0915 3      THEN
830 0916 4      BEGIN
831 0917 4
832 0918 4      IF .blocks EQL 0      | Block=0 means look for the biggest free space
833 0919 4      THEN
834 0920 5      BEGIN
835 0921 5      IF .len GTRU .mat_len      | Bigger than what we've seen so far
836 0922 5      THEN
837 0923 6      BEGIN
838 0924 6
839 0925 6      | Save the entry as a match. Note that all 4 saves can be collapsed by the optimizer.
840 0926 6
841 0927 6      mat_len = .len;      | Save the length of this entry,
842 0928 6      mat_ent = .cur;      | its address,
843 0929 6      mat_seg = .ctx [rt11ctx$l_seg_number]; | its segment number,
844 0930 6      mat_blk = .ctx [rt11ctx$l_start_block]; | and the pbn where the free space starts
845 0931 5      END;
846 0932 5      END
847 0933 4      ELSE      | Block<>0 means look for a close match
848 0934 5      BEGIN
849 0935 5      IF .len EQL .blocks      | Exactly what we are looking for
850 0936 5      THEN
851 0937 6      BEGIN
852 0938 6
853 0939 6      | Save the entry as a match. Note that all 4 saves can be collapsed by the optimizer.
854 0940 6
855 0941 6      exact_match = true;      | Set the flag to stop scanning
856 0942 6      mat_len = .len;      | Save the length of this entry,
857 0943 6      mat_ent = .cur;      | its address,
858 0944 6      mat_seg = .ctx [rt11ctx$l_seg_number]; | its segment number,
859 0945 6      mat_blk = .ctx [rt11ctx$l_start_block]; | and the pbn where the free space starts
860 0946 6      END
861 0947 7      ELSE IF ((.len LSSU .mat_len)      | Closer to correct than the previous match
862 0948 6      OR (.mat_len EQL 0))      | or no previous match
863 0949 5      THEN
864 0950 6      BEGIN
865 0951 6
866 0952 6      | Save the entry as a match. Note that all 4 saves can be collapsed by the optimizer.
867 0953 6
868 0954 6      mat_len = .len;      | Save the length of this entry,
869 0955 6      mat_ent = .cur;      | its address,
870 0956 6      mat_seg = .ctx [rt11ctx$l_seg_number]; | its segment number,
871 0957 6      mat_blk = .ctx [rt11ctx$l_start_block]; | and the pbn where the free space starts
```

```

: 872 0958 5
: 873 0959 4
: 874 0960 3
: 875 0961 2
: 876 0962 1
: 877 0963 0
: 878 0964 0
: 879 0965 0
: 880 0966 2

                                END;
                                END;
                                END;
: If we have seen an exact match we can stop scanning this segment
:
IF .exact_match THEN EXITLOOP;

END;
```



```

: 882 0967 2 ! Now put a zero-length entry after the matched entry and set the return values
: 883 0968 2
: 884 0969 2 IF .mat_len NEQ 0 . Means that we found one
: 885 0970 2 THEN
: 886 0971 2 BEGIN
: 887 0972 2 LOCAL
: 888 0973 2 eos : $ref_bblock, ! End of segment entry
: 889 0974 2 emp : $ref_bblock, ! Entry to be turned into a zero-length empty
: 890 0975 2 sl; ! Length of directory to slide up one entry
: 891 0976 2
: 892 0977 2 ! Get a pointer to the segment containing the match, return if error
: 893 0978 2
: 894 0979 2 seg = exch$rt11_dirseg_get (.volb, .mat_seg);
: 895 0980 2 $logic_check (2, (.seg NEQ 0), 155);
: 896 0981 2 ent_len = rt11ent$k_length + .seg [rt11hdr$w_extra_bytes];
: 897 0982 2
: 898 0983 2 ! Find the end of the segment. We verified that one exists in the first scan.
: 899 0984 2
: 900 0985 2 eos = .mat_ent; ! Point to the matched entry
: 901 0986 2 WHILE 1
: 902 0987 2 DO
: 903 0988 2 BEGIN
: 904 0989 2 eos = .eos + .ent_len; ! Advance to the next entry
: 905 0990 2 $logic_check (1, (.eos LSSU (.seg + rt11$k_dirseglen)), 144); ! It can't loop forever, can it?
: 906 0991 2 IF .eos [rt11ent$v_type] EQL rt11ent$m_typ_end_segment
: 907 0992 2 THEN
: 908 0993 2 EXITLOOP;
: 909 0994 2 END;
: 910 0995 2
: 911 0996 2 ! If we are looking for a specific block, we might need to split the current empty into two entries, the
: 912 0997 2 second of which will start on the requested block.
: 913 0998 2
: 914 0999 2 IF .start NEQ 0 ! /START_BLOCK has been requested
: 915 1000 2 AND
: 916 1001 2 .start NEQ .mat_blk ! We are lucky, it already begins on exactly the right block
: 917 1002 2 THEN
: 918 1003 2 BEGIN
: 919 1004 2
: 920 1005 2 ! Make sure that there is room to add two more entries to this segment. If not, signal a full direc
: 921 1006 2
: 922 1007 2 IF ((.eos+2 + (2*.ent_len)) GEQU (.seg + rt11$k_dirseglen))
: 923 1008 2 THEN
: 924 1009 2 BEGIN
: 925 1010 2 IF .recursive ! If we have already tried it a seco
: 926 1011 2 THEN ! then cleaning didn't help and we
: 927 1012 2 RETURN exch$_rt11_overflow ! return an error. Otherwise, call
: 928 1013 2 ELSE ! routine again with a dummy parame
: 929 1014 2 RETURN exch$rtacp_find_empty_area (.ctx, .blocks, .start, 0); ! flag the call as recursive
: 930 1015 2 END;
: 931 1016 2
: 932 1017 2 ! Slide the rest of the segment up one entry and split the empty entry
: 933 1018 2
: 934 1019 2 emp = .mat_ent; ! Point to the matched entry (will become an
: 935 1020 2 sl = .eos+2 - .emp; ! Length of segment between empty and end
: 936 1021 2 CHSMOVE (.sl, .emp, .emp + .ent_len); ! Slide rest of segment up
: 937 1022 2 mat_ent = .mat_ent + .ent_len; ! Move the pointer to the matched entry
: 938 1023 2 eos = .eos + .ent_len; ! Move the pointer to the end of the segment

```

```

939      1024 4      CH$FILL (0, rt1lent$k_length, .emp);           ! Set all fields of empty entry to null
940      1025 4      emp [rt1lent$b_type_byte] = rt1lent$m_typ_empty; ! Set to an empty entry
941      1026 4      sl = .start - .mat_blk;                       ! Get number of blocks to put in the empty
942      1027 4      emp [rt1lent$w_blocks] = .sl;                 ! Set the size of the new empty
943      1028 4      mat_ent [rt1lent$w_blocks] = .mat_ent [rt1lent$w_blocks] - .sl; ! Subtract the blocks from the match
944      1029 4      mat_blk = .mat_blk + .sl;                     ! Move the starting block
945      1030 4
946      1031 3      END;
947      1032 3
948      1033 3      ! Make sure that there is room to add one more entry to this segment. If not, signal a full directory
949      1034 3      !
950      1035 4      IF ((.eos+2 + .ent_len) GEQU (.seg + rt11$k_dirseglen))
951      1036 3      THEN
952      1037 4      BEGIN
953      1038 4      IF .recursive                                     ! If we have already tried it a seco
954      1039 4      THEN                                           ! then cleaning didn't help and we
955      1040 4      status = exch$_rt11_overflow                    ! return an error. Otherwise, call
956      1041 4      ELSE                                           ! routine again with a dummy parame
957      1042 4      RETURN exch$rtacp_find_empty_area (.ctx, .blocks, .start, 0); ! flag the call as recursive
958      1043 4      END
959      1044 4
960      1045 3      ELSE
961      1046 4      BEGIN
962      1047 4
963      1048 4      ! Slide the rest of the segment up one entry and make the empty entry,
964      1049 4      !
965      1050 4      emp = .mat_ent + .ent_len;                       ! Point to one past the matched entry
966      1051 4      sl = .eos+2 - .emp;                             ! Length of segment between empty and end
967      1052 4      CH$MOVE (.sl, .emp, .emp + .ent_len);          ! Slide rest of segment up
968      1053 4      CH$FILL (0, rt1lent$k_length, .emp);          ! Set all fields of entry to null
969      1054 4      emp [rt1lent$b_type_byte] = rt1lent$m_typ_empty; ! Zap whole byte to clear protect bit
970      1055 4
971      1056 4      ! Put the info in the context block
972      1057 4      !
973      1058 4      ctx [rt11ctx$l_start_block] = .mat_blk;         ! Copy the pbn where the file starts
974      1059 4      ctx [rt11ctx$l_seg_number] = .mat_seg;         ! The segment number
975      1060 4      ctx [rt11ctx$a_seg_address] = .seg;            ! The address of the start of the se
976      1061 4      ctx [rt11ctx$a_ent_address] = .mat_ent;       ! The address of the entry
977      1062 4      CH$MOVE (rt11ctx$s_entry, .mat_ent, ctx [rt11ctx$t_entry]); ! And the entry itself
978      1063 4
979      1064 4      exch$rt11_dirseg_put (.volb, .mat_seg);        ! Save the changes
980      1065 4      status = true;
981      1066 3      END;
982      1067 3      END
983      1068 3
984      1069 3      ! Either no free space at all (.blocks = 0) or none as large as requested
985      1070 3      !
986      1071 2      ELSE
987      1072 3      BEGIN
988      1073 3      IF .start NEQ 0                                     ! If specific start block was requested, then it was past th
989      1074 3      THEN                                           ! end of the volume (or contained in last entry in director
990      1075 3      status = exch$_stnotavail
991      1076 3      ELSE IF .blocks EQ 0                             ! If blocks requested is 0, then there was no space anyplace
992      1077 3      THEN                                           ! we should return a volume full error
993      1078 3      status = exch$_volume_full
994      1079 3      ELSE
995      1080 4      BEGIN

```

```

: 996      1081  4      IF .recursive
: 997      1082  4      THEN
: 998      1083  4      status = exch$_volume_full
: 999      1084  4      ELSE
1000      1085  4      RETURN exch$rtacp_find_empty_area (.ctx, .blocks, .start, 0);
1001      1086  3      END;
1002      1087  2      END;
1003      1088  2
1004      1089  2  $logic_check (2, (exch$rtacp_verify_directory (.volb)), 192);
1005      1090  2
1006      1091  2  RETURN .status;
: 1007      1092  1  END;

```

```

: If we have already tried it a seco
: then cleaning didn't help and we
: return an error. Otherwise, call
: routine again with a dummy parame
: flag the call as recursive

```

```

      OFFC 00000
      SE      20  C2 00002
      59      04  AC  D0 00005
      52 008200F4 8F  D0 00009
      51      0217 8F  3C 00010
      50      59  D0 00015
      04 AE      14  A9  D0 00018
      52 041B00F3 8F  D0 00023
      51      0212 8F  3C 0002A
      50      04  AE  D0 0002F
      00000000G EF  16 00033
      7E      A9  D4 00039
      6E      76  A9  9E 0003C
      00      BE  D4 00040
      53      D4 00043
      04      6C  91 00045
      06      12 00048
      1C AE      01  D0 0004A
      03      11 0004E
      1C AE      D4 00050 1$:
      1C AE      DD 00053 2$:
      08 AE      DD 00056
      FE12 CF      02  FB 00059
      OC AE      50  D0 0005E
      03      OC AE  E8 00062
      01DA 31 00066
      54      D4 00069 3$:
      55      32  D0 0006B
      55      DD 0006E 4$:
      59      DD 00070
      0000V CF      02  FB 00072
      50      D5 00077
      5E      13 00079
      51      08  A0  3C 0007B
      OC      AC  D5 0007F
      1C      13 00082
      .EXTRN EXCH$_STNOTAVAIL
      .EXTRN EXCH$_RT11_OVERFLOW
      .EXTRN EXCH$_VOLUME_FULL
      .ENTRY EXCH$RTACP_FIND_EMPTY_AREA, Save R2,R3,R4,- : 0787
      R5,R6,R7,R8,R9,R10,R11
      SUBL2 #32, SP
      MOVL CTX, R9 : 0847
      MOVL #8519924, R2 : 0857
      MOVZWL #535, R1
      MOVL R9, R0
      JSB EXCH$UTIL_BLOCK_CHECK
      MOVL 20(R9), 4(SP) : 0858
      MOVL #68878579, R2
      MOVZWL #530, R1
      MOVL 4(SP), R0
      JSB EXCH$UTIL_BLOCK_CHECK
      CLRL 126(R9) : 0862
      MOVAB 118(R9), (SP) : 0863
      CLRL @0(SP)
      CLRL MAT_LEN : 0864
      CMPB (APT), #4 : 0865
      BNEQ 1$
      MOVL #1, RECURSIVE
      BRB 2$
      CLRL RECURSIVE
      PUSHL RECURSIVE : 0870
      PUSHL 8(SP)
      CALLS #2, EXCH$RTACP_CONSOLIDATE
      MOVL R0, STATUS
      BLBS STATUS, 3$
      BRW 26$
      CLRL EXACT_MATCH : 0876
      MOVL #50, FLAGS : 0877
      PUSHL FLAGS : 0878
      PUSHL R9
      CALLS #2, EXCH$RTACP_NEXT_ENTRY
      TSTL CUR
      BEQL 12$
      MOVZWL 8(CUR), LEN : 0884
      TSTL START : 0888
      BEQL 6$

```













```

: 1066 1149 1 GLOBAL ROUTINE exch$rtacp_next_entry (ctx : $ref_bblock, flags : $bblock) = %SBTTL 'exch$rtacp_next_entr
: 1067 1150 2 BEGIN
: 1068 1151 2 +-
: 1069 1152 2
: 1070 1153 2 FUNCTIONAL DESCRIPTION:
: 1071 1154 2
: 1072 1155 2 Return the next entry from an RT-11 directory
: 1073 1156 2
: 1074 1157 2 INPUT:
: 1075 1158 2
: 1076 1159 2 ctx - pointer to an RT11CTX structure
: 1077 1160 2 flags - a structure of bits with the following meanings:
: 1078 1161 2
: 1079 1162 2 flags [rtxt$v_permanent] - return the address of the next permanent entry
: 1080 1163 2 flags [rtxt$v_empty] - return the address of the next empty entry
: 1081 1164 2 flags [rtxt$v_tentative] - return the address of the next tentative entry
: 1082 1165 2 flags [rtxt$v_unknown] - return the address of the next entry with invalid
: 1083 1166 2 flags [rtxt$v_skip_check] - skip the check for a moved entry
: 1084 1167 2 flags [rtxt$v_skip_expand] - skip expanding the radix-50 name to ascii
: 1085 1168 2
: 1086 1169 2 More than one flag can be set if more than one type of entry is desired. The end-of-segment entry can not
: 1087 1170 2 returned, but the caller can examine the [rt11ctx$l_seg_number] and interpret a change as an implicit eos.
: 1088 1171 2
: 1089 1172 2 IMPLICIT INPUTS:
: 1090 1173 2
: 1091 1174 2 anything we can find hanging off the context block
: 1092 1175 2
: 1093 1176 2 OUTPUTS:
: 1094 1177 2
: 1095 1178 2 none
: 1096 1179 2
: 1097 1180 2 IMPLICIT OUTPUTS:
: 1098 1181 2
: 1099 1182 2 the context block is updated with context necessary for wildcard processing
: 1100 1183 2
: 1101 1184 2 ROUTINE VALUE:
: 1102 1185 2
: 1103 1186 2 address of the next entry, 0 if no more entries of the desired type
: 1104 1187 2
: 1105 1188 2 SIDE EFFECTS:
: 1106 1189 2
: 1107 1190 2 none
: 1108 1191 2 --
: 1109 1192 2
: 1110 1193 2 $dbgtrc_prefix ('rtacp_next_entry> ');
: 1111 1194 2
: 1112 1195 2 LOCAL
: 1113 1196 2 get_dirseg,
: 1114 1197 2 start_block,
: 1115 1198 2 seg_num,
: 1116 1199 2 seg : $ref_bblock, ! a pointer to the current directory segment
: 1117 1200 2 ent : $ref_bblock ! a pointer to the current directory entry
: 1118 1201 2 ;
: 1119 1202 2
: 1120 1203 2 BIND
: 1121 1204 2 volb = ctx [rt11ctx$a_assoc_volb] : $ref_bblock
: 1122 1205 2 ;

```

```
: 1124      1206      2 $debug_print_fao ('entry - ctx !XL', .ctx);
: 1125      1207      2 $block_check (2, .ctx, rt11ctx, 552);
: 1126      1208      2 $block_check (2, .volb, volb, 553);
: 1127      1209      2 $logic_check (5, (exch$rtacp_verify_directory (.volb)), 212);
: 1128      1210      2
: 1129      1211      2 ! If the context segment number is null, then we are starting from scratch to find the file
: 1130      1212      2
: 1131      1213      2 IF .ctx [rt11ctx$L_seg_number] EQL 0
: 1132      1214      2 THEN
: 1133      1215      2 BEGIN
: 1134      1216      2     ! Start with the first directory segment
: 1135      1217      2     !
: 1136      1218      2     seg_num = 1;
: 1137      1219      2     !
: 1138      1220      2     ! Set flag that we must read a directory segment pointer
: 1139      1221      2     !
: 1140      1222      2     get_dirseg = true;
: 1141      1223      2     !
: 1142      1224      2     !
: 1143      1225      2     END
: 1144      1226      2
: 1145      1227      2 ! If non-null, we are doing a subsequent lookup in a wildcard search
: 1146      1228      2
: 1147      1229      2 ELSE
: 1148      1230      2 BEGIN
: 1149      1231      2     !
: 1150      1232      2     ! Check that the directory entry positions are still good. The entry might have moved - if so repositio
: 1151      1233      2     ! the new location. The skip_check option cannot be used if there is any possibility of any directory
: 1152      1234      2     ! modifications between calls.
: 1153      1235      2     !
: 1154      1236      2     IF NOT (.flags [rt11ctx$v_skip_check])
: 1155      1237      2     THEN
: 1156      1238      2         exch$rtacp_check_position (.ctx);
: 1157      1239      2     !
: 1158      1240      2     ! Get the segment, entry, and start block, and adjust to point to the next entry
: 1159      1241      2     !
: 1160      1242      2     seg = .ctx [rt11ctx$a_seg_address];           ! Get our seg from the context block
: 1161      1243      2     ent = .ctx [rt11ctx$a_ent_address];           ! Get the entry from the context block too
: 1162      1244      2     seg_num = .ctx [rt11ctx$L_seg_number];         ! Get the segment number
: 1163      1245      2     start_block =                                ! Add length of current entry
: 1164      1246      2         .ctx [rt11ctx$L_start_block] + .ent [rt11ent$w_blocks];
: 1165      1247      2     ent = .ent + rt11ent$w_length                    ! Skip the pointer to the next entry
: 1166      1248      2         + .seg [rt11hdr$w_extra_bytes];
: 1167      1249      2     ctx [rt11ctx$a_ent_address] = .ent;           ! Save it in the context block too
: 1168      1250      2     !
: 1169      1251      2     ! Clear the flag, we don't need to read a directory segment pointer right now
: 1170      1252      2     !
: 1171      1253      2     get_dirseg = false;
: 1172      1254      2     !
: 1173      1255      2 END;
```

```
1175 1256 2 ! Loop through the directory looking for this file
1176 1257 2
1177 1258 2 WHILE .seg_num NEQ 0
1178 1259 2 DO
1179 1260 2 BEGIN
1180 1261 2
1181 1262 2 IF .get_dirseg
1182 1263 2 THEN
1183 1264 2 BEGIN
1184 1265 2
1185 1266 2 ! Get a pointer to the current segment
1186 1267 2 !
1187 1268 2 seg = exch$rt11_dirseg_get (.volb, .seg_num);
1188 1269 2 $logic_check (2, (.seg NEQ 0), 156);
1189 1270 2 start_block = .seg [rt11hdr$w_start_block]; ! Fetch the pbn where the first file in this segment
1190 1271 2 ent = .seg + rt11hdr$k_length; ! Get a pointer to the first directory entry
1191 1272 2
1192 1273 2 END;
1193 1274 2
1194 1275 2 get_dirseg = true; ! We always want to get additional segs
1195 1276 2
1196 1277 2 WHILE (.ent LSSU (.seg + rt11$k_dirseglen))
1197 1278 2 DO
1198 1279 2 BEGIN
1199 1280 2
1200 1281 2 ! Process the entry depending on its type
1201 1282 2 !
1202 1283 2 CASE .ent [rt11ent$v_type] FROM 0 TO rt11ent$m_typ_end_segment OF
1203 1284 2 SET
1204 1285 2 [rt11ent$m_typ_permanent] :
1205 1286 2
1206 1287 2 BEGIN
1207 1288 2
1208 1289 2 IF .flags [rt11ent$v_permanent]
1209 1290 2 THEN
1210 1291 2 BEGIN
1211 1292 2
1212 1293 2 ! Copy the standard portion of the entry to the context block
1213 1294 2 !
1214 1295 2 CH$MOVE (rt11ent$k_length, .ent, ctx [rt11ctx$t_entry]);
1215 1296 2
1216 1297 2 ! Expand the directory entry filename information into the context block
1217 1298 2 !
1218 1299 2 IF NOT (.flags [rt11ent$v_skip_expand])
1219 1300 2 THEN
1220 1301 2 exch$rt11_expand_filename (.ctx);
1221 1302 2
1222 1303 2 $debug_print_fao ('Found '!AF!AF'', .volb [volb$l_vol_ident_len], volb [volb$t_vol_ident
1223 1304 2 .ctx [rt11ctx$l_exp_fullname_len], ctx [rt11ctx$t_exp_fullna
1224 1305 2
1225 1306 2 ! Save the directory position in the context block
1226 1307 2 !
1227 1308 2 ctx [rt11ctx$l_start_block] = .start_block;
1228 1309 2 ctx [rt11ctx$l_seg_number] = .seg_num;
1229 1310 2 ctx [rt11ctx$a_seg_address] = .seg;
1230 1311 2 ctx [rt11ctx$a_ent_address] = .ent;
1231 1312 2
```

```
: 1232 1313 6
: 1233 1314 6          RETURN .ent;
: 1234 1315 5          END;
: 1235 1316 5
: 1236 1317 5          start_block = .start_block + .ent [rt11ent$w_blocks];
: 1237 1318 5
: 1238 1319 4          END;
: 1239 1320 4
: 1240 1321 4          [rt11ent$m_typ_tentative] :
: 1241 1322 4
: 1242 1323 5          BEGIN
: 1243 1324 5
: 1244 1325 5          IF .flags [rtxnt$v_tentative]
: 1245 1326 5          THEN
: 1246 1327 6              BEGIN
: 1247 1328 6                  ! Copy the standard portion of the entry to the context block
: 1248 1329 6                  !
: 1249 1330 6                  CH$MOVE (rt11ent$k_length, .ent, ctx [rt11ctx$t_entry]);
: 1250 1331 6                  !
: 1251 1332 6                  ! Expand the directory entry filename information into the context block
: 1252 1333 6                  !
: 1253 1334 6                  !
: 1254 1335 7                  IF NOT (.flags [rtxnt$v_skip_expand])
: 1255 1336 6                  THEN
: 1256 1337 6                      exch$rt11_expand_filename (.ctx);
: 1257 1338 6
: 1258 P 1339 6          $debug_print_fao ('Found '!AF!AF'', .volb [volb$l_vol_ident_len], volb [volb$t_vol_ident
: 1259 1340 6                      .ctx [rt11ctx$l_exp_fullname_len], ctx [rt11ctx$t_exp_fullna
: 1260 1341 6
: 1261 1342 6          ! Save the directory position in the context block
: 1262 1343 6          !
: 1263 1344 6          ctx [rt11ctx$l_start_block] = .start_block;
: 1264 1345 6          ctx [rt11ctx$l_seg_number] = .seg_num;
: 1265 1346 6          ctx [rt11ctx$a_seg_address] = .seg;
: 1266 1347 6          ctx [rt11ctx$a_ent_address] = .ent;
: 1267 1348 6
: 1268 1349 6          RETURN .ent;
: 1269 1350 5          END;
: 1270 1351 5
: 1271 1352 5          start_block = .start_block + .ent [rt11ent$w_blocks];
: 1272 1353 5
: 1273 1354 4          END;
: 1274 1355 4
: 1275 1356 4          [rt11ent$m_typ_empty] :
: 1276 1357 4
: 1277 1358 5          BEGIN
: 1278 1359 5
: 1279 1360 5          IF .flags [rtxnt$v_empty]
: 1280 1361 5          THEN
: 1281 1362 6              BEGIN
: 1282 1363 6                  ! Copy the standard portion of the entry to the context block
: 1283 1364 6                  !
: 1284 1365 6                  !
: 1285 1366 6                  CH$MOVE (rt11ent$k_length, .ent, ctx [rt11ctx$t_entry]);
: 1286 1367 6                  !
: 1287 1368 6                  ! Expand the directory entry filename information into the context block
: 1288 1369 6                  !
```



```

: 1289      1370 7      IF NOT (.flags [rtxt$v_skip_expand])
: 1290      1371 6      THEN
: 1291      1372 6          exch$rt11_expand_filename (.ctx);
: 1292      1373 6
: 1293      1374 6      P $debug_print_fao ('found '!AF!AF'', .volb [volb$l_vol_ident_len], volb [volb$t_vol_ident
: 1294      1375 6          .ctx [rt11ctx$l_exp_fullname_len], ctx [rt11ctx$t_exp_fullna
: 1295      1376 6
: 1296      1377 6      ! Save the directory position in the context block
: 1297      1378 6      !
: 1298      1379 6      ctx [rt11ctx$l_start_block] = .start_block;
: 1299      1380 6      ctx [rt11ctx$l_seg_number] = .seg_num;
: 1300      1381 6      ctx [rt11ctx$a_seg_address] = .seg;
: 1301      1382 6      ctx [rt11ctx$a_ent_address] = .ent;
: 1302      1383 6
: 1303      1384 6      RETURN .ent;
: 1304      1385 5      END;
: 1305      1386 5
: 1306      1387 5      start_block = .start_block + .ent [rt11ent$w_blocks];
: 1307      1388 5
: 1308      1389 4      END;
: 1309      1390 4
: 1310      1391 4      [INRANGE, OTRANGE] :
: 1311      1392 4
: 1312      1393 5      BEGIN
: 1313      1394 5
: 1314      1395 5      IF .flags [rtxt$v_unknown]
: 1315      1396 5      THEN
: 1316      1397 6          BEGIN
: 1317      1398 6
: 1318      1399 6          ! Copy the standard portion of the entry to the context block
: 1319      1400 6          !
: 1320      1401 6          CH$MOVE (rt11ent$k_length, .ent, ctx [rt11ctx$t_entry]);
: 1321      1402 6
: 1322      1403 6          ! Expand the directory entry filename information into the context block
: 1323      1404 6          !
: 1324      1405 7          IF NOT (.flags [rtxt$v_skip_expand])
: 1325      1406 6          THEN
: 1326      1407 6              exch$rt11_expand_filename (.ctx);
: 1327      1408 6
: 1328      1409 6      P $debug_print_fao ('found '!AF!AF'', .volb [volb$l_vol_ident_len], volb [volb$t_vol_ident
: 1329      1410 6          .ctx [rt11ctx$l_exp_fullname_len], ctx [rt11ctx$t_exp_fullna
: 1330      1411 6
: 1331      1412 6          ! Save the directory position in the context block
: 1332      1413 6          !
: 1333      1414 6          ctx [rt11ctx$l_start_block] = .start_block;
: 1334      1415 6          ctx [rt11ctx$l_seg_number] = .seg_num;
: 1335      1416 6          ctx [rt11ctx$a_seg_address] = .seg;
: 1336      1417 6          ctx [rt11ctx$a_ent_address] = .ent;
: 1337      1418 6
: 1338      1419 6          RETURN .ent;
: 1339      1420 5          END;
: 1340      1421 5
: 1341      1422 5      start_block = .start_block + .ent [rt11ent$w_blocks];
: 1342      1423 5
: 1343      1424 4      END;
: 1344      1425 4
: 1345      1426 4      [rt11ent$m_typ_end_segment] :
```

```

: 1346      1427  4
: 1347      1428  4
: 1348      1429  4
: 1349      1430  4
: 1350      1431  4
: 1351      1432  4
: 1352      1433  4
: 1353      1434  4
: 1354      1435  4
: 1355      1436  4
: 1356      1437  4
: 1357      1438  4
: 1358      1439  4
: 1359      1440  4
: 1360      1441  4
: 1361      1442  4
: 1362      1443  4
: 1363      1444  4
: 1364      1445  4
: 1365      1446  4
: 1366      1447  4
: 1367      1448  4
: 1368      1449  4

EXITLOOP:
TES;
! Skip to the next entry
!
ent = .ent + rt1lent$k_length + .seg [rt11hdr$w_extra_bytes];
END;
! Skip to the next segment
!
seg_num = .seg [rt11hdr$w_next_seg];
END;
! Mark the context with an invalid segment number to prevent recycling through the directory,
!
ctx [rt11ctx$l_seg_number] = -1;
RETURN 0;
END;

```

		OFFC	00000	.ENTRY	EXCH\$RTACP NEXT ENTRY, Save R2,R3,R4,R5,R6,-;	
		5E	0C C2 00002	SUBL2	#12, SP	1149
		57	04 AC D0 00005	MOVL	CTX, R7	1204
		52	008200F4 8F D0 00009	MOVL	#8519924, R2	1207
		51	0228 8F 3C 00010	MOVZWL	#552, R1	
		50	57 D0 00015	MOVL	R7, R0	
			00000000G EF 16 00018	JSB	EXCH\$UTIL_BLOCK_CHECK	
		52	041B00F3 8F D0 0001E	MOVL	#68878579, R2	1208
		51	0229 8F 3C 00025	MOVZWL	#553, R1	
		50	14 A7 D0 0002A	MOVL	20(R7), R0	
			00000000G EF 16 0002E	JSB	EXCH\$UTIL_BLOCK_CHECK	
		5B	76 A7 9E 00034	MOVAB	118(R7), R11	1213
			6B D5 00038	TSTL	(R11)	
			08 12 0003A	BNEQ	1\$	
		5A	01 D0 0003C	MOVL	#1, SEG_NUM	1219
		6E	01 D0 0003F	MOVL	#1, GET_DIRSEG	1223
			2E 11 00042	BRB	3\$	1213
07	08	AC	04 E0 00044	BBS	#4, FLAGS, 2\$	1236
			57 DD 00049	PUSHL	R7	1238
	F8F7	CF	01 FB 0004B	CALLS	#1, EXCH\$RTACP_CHECK_POSITION	
		58	7A A7 D0 00050	MOVL	122(R7), SEG	1242
		56	7E A7 D0 00054	MOVL	126(R7), ENT	1243
		5A	6B D0 00058	MOVL	(R11), SEG_NUM	1244
		59	08 A6 3C 0005B	MOVZWL	8(ENT), START_BLOCK	1246
		59	72 A7 C0 0005F	ADDL2	114(R7), START_BLOCK	
		50	06 A8 3C 00063	MOVZWL	6(SEG), R0	1248
		56	0E A046 9E 00067	MOVAB	14(R0)(ENT), ENT	
	7E	A7	56 D0 0006C	MOVL	ENT, 126(R7)	1249

				6E	D4	00070		CLRL	GET_DIRSEG		1253
				5A	D5	00072	3\$:	TSTL	SEG_NUM		1258
				03	12	00074		BNEQ	4\$		
				00C1	31	00076		BRW	18\$		
		2C		6E	E9	00079	4\$:	BLBC	GET DIRSF,, 6\$		1262
				5A	DD	0007C		PUSHL	SEG_NUM		1268
			14	A7	DD	0007E		PUSHL	20(R7)		
	00000000G	EF		02	FB	00081		CALLS	#2, EXCH\$RT11_DIRSEG_GET		
		58		50	DD	00088		MOVL	R0, SEG		
				13	12	0008B		BNEQ	5\$		1269
		7E		9C	8F	9A	0008D	MOVZBL	#156, -(SP)		
				01	DD	00091		PUSHL	#1		
	00000000G	00	00000000G	8F	DD	00093		PUSHL	#EXCH\$ BADLOGIC		
		59		03	FB	00099		CALLS	#3, LIB\$STOP		
		56		08	A8	3C	000A0	MOVZWL	8(SEG), START_BLOCK		1270
		6E		0A	A8	9E	000A4	MOVAB	10(R8), ENT		1271
		08		01	DD	000A8	6\$:	MOVL	#1, GET DIRSEG		1275
		04		0400	C8	9E	000AB	MOVAB	1024(R8), 8(SP)		1277
		08		06	A8	9E	000B1	MOVAB	6(SEG), 4(SP)		1434
					56	D1	000B6	7\$:	CMLP	ENT, 8(SP)	1277
					77	1E	000BA	BGEQU	17\$		
7E	01	A6		04	00	EF	000BC	EXTZV	#0, #4, 1(ENT), -(SP)		1283
		08		00	8E	CF	000C2	CASEL	(SP)+, #0, #8		
0012		0030		0029	0012		000C6	8\$:	.WORD		
0012		0012		0012	0023		000CE		9\$-8\$,-		
					006D		000D6		12\$-8\$,-		
									13\$-8\$,-		
									9\$-8\$,-		
									11\$-8\$,-		
									9\$-8\$,-		
									9\$-8\$,-		
									9\$-8\$,-		
									17\$-8\$		
									#3, FLAGS, 16\$		1395
									#14, (ENT), 56(R7)		1401
									#5, FLAGS, 14\$		1405
									15\$		1414
									BRB		1290
									BLBC	FLAGS, 16\$	1296
									BRB	10\$	1325
									BBC	#2, FLAGS, 16\$	1331
									BRB	10\$	1360
									BBC	#1, FLAGS, 16\$	1366
									MOVZWL	#14, (ENT), 56(R7)	1370
									BBS	#5, FLAGS, 15\$	1372
									R7		
									PUSHL	R7	
									CALLS	#1, EXCH\$RT11_EXPAND_FILENAME	
									MOVL	START_BLOCK, T14(R7)	1379
									MOVL	SEG_NUM, (R11)	1380
									MOVL	SEG, 122(R7)	1381
									MOVL	ENT, 126(R7)	1382
									MOVL	ENT, R0	1384
									RET		
									MOVZWL	8(ENT), R0	1387
									ADDL2	R0, START_BLOCK	
									MOVZWL	24(SP), R0	1434
									MOVAB	14(R0)[ENT], ENT	
									BRB	7\$	1277
									MOVZWL	2(SEG), SEG_NUM	1440

EXCH\$RTACP  
V04-000

RT11 directory routines  
exch\$rtacp\_next\_entry (ctx, flags)

N 6  
16-Sep-1984 01:19:05  
14-Sep-1984 12:29:08

VAX-11 Bliss-32 V4.0-742  
[EXCHNG.SRC]EXCRTACP.B32;1

Page 42  
(16)

EXC  
V04

6B

FF38	31	00137	
01	CE	0013A	18\$:
50	D4	0013D	
	04	0013F	

BRW	3\$	
MNEGL	#1,	(R11)
CLRL	RO	
RET		

: 1258  
: 1446  
: 1448  
: 1449

; Routine Size: 320 bytes,      Routine Base: EXCH\$RTACP\_CODE + 06B9

.....

```
: 1370 1450 1 GLOBAL ROUTINE exch$rtacp_verify_directory (volb : $ref_bblock) = %SBTTL 'exch$rtacp_verify_directory
: 1371 1451 2 BEGIN
: 1372 1452 2 ++
: 1373 1453 2
: 1374 1454 2 FUNCTIONAL DESCRIPTION:
: 1375 1455 2
: 1376 1456 2 Traverse the directory and check it for validity. We also count the number of blocks.
: 1377 1457 2
: 1378 1458 2 INPUTS:
: 1379 1459 2
: 1380 1460 2 volb - pointer to volb which has been connected to the RT-11 device
: 1381 1461 2
: 1382 1462 2 IMPLICIT INPUTS:
: 1383 1463 2
: 1384 1464 2 none
: 1385 1465 2
: 1386 1466 2 OUTPUTS:
: 1387 1467 2
: 1388 1468 2 blocks - address of longword to receive number of blocks
: 1389 1469 2
: 1390 1470 2 IMPLICIT OUTPUTS:
: 1391 1471 2
: 1392 1472 2 none
: 1393 1473 2
: 1394 1474 2 ROUTINE VALUE:
: 1395 1475 2
: 1396 1476 2 true if valid, false if not
: 1397 1477 2
: 1398 1478 2 SIDE EFFECTS:
: 1399 1479 2
: 1400 1480 2 error conditions will be signaled
: 1401 1481 2 --
: 1402 1482 2
: 1403 1483 2 $dbgtrc_prefix ('exch$rtacp_verify_directory> ');
: 1404 1484 2
: 1405 1485 2 LOCAL
: 1406 1486 2 rtv : $ref_bblock, ! a pointer to the rt11 volb extension
: 1407 1487 2 seg : $ref_bblock, ! a pointer to the current directory segment
: 1408 1488 2 ent : $ref_bblock, ! a pointer to the current directory entry
: 1409 1489 2 sum_blocks, ! running block count
: 1410 1490 2 unknowns, ! count of unknown directory entries
: 1411 1491 2 seg_num,
: 1412 1492 2 end_segment_seen,
: 1413 1493 2 missing_end,
: 1414 1494 2 status
: 1415 1495 2 ;
: 1416 1496 2
: 1417 1497 2 $debug_print_lit ('entry');
: 1418 1498 2
: 1419 1499 2 $block_check (2, .volb, volb, 470);
: 1420 1500 2
: 1421 1501 2 ! Assume that we will find a bad directory
: 1422 1502 2 !
: 1423 1503 2 status = 0;
: 1424 1504 2 unknowns = false; ! Clear the unknown entry flag
: 1425 1505 2 missing_end = 0;
```

```
1427 1506 2 ! Get the pointer to our volb extension and to the root segment
1428 1507 2
1429 1508 2 rtv = .volb [volb$a_vfmt_specific];
1430 1509 2 $block_check (2, .rtv, rt11, 471);
1431 1510 2 seg = rtv [rt11$t_block_0] + (512 * rt11$k_root_block);
1432 1511 2
1433 1512 2 ! Start with the first directory segment
1434 1513 2
1435 1514 2 seg_num = 1;
1436 1515 2
1437 1516 2 ! Assume that the files will start after the last segment
1438 1517 2
1439 1518 2 sum_blocks = rt11$k_root_block + (2 * .seg [rt11hdr$w_num_segs]);
1440 1519 2 $debug_print_fao ('number of segments !UL', .seg [rt11hdr$w_num_segs]);
1441 1520 2
1442 1521 2 ! Loop through all the segments in the directory
1443 1522 2
1444 1523 2 WHILE .seg_num NEQ 0
1445 1524 2 DO
1446 1525 2 BEGIN
1447 1526 2
1448 1527 2 ! Get a pointer to the current segment
1449 1528 2
1450 1529 2 seg = exch$rt11_dirseg_get (.volb, .seg_num);
1451 1530 2 IF .seg EQL 0
1452 1531 2 THEN
1453 1532 2 RETURN false;
1454 1533 2
1455 1534 2 ! If the current segment start block is not what we expect, signal and use new value
1456 1535 2
1457 1536 2 $debug_print_fao ('expected block !UL, actual block !UL', .sum_blocks, .seg [rt11hdr$w_start_block]);
1458 1537 2 IF (.sum_blocks NEQ .seg [rt11hdr$w_start_block])
1459 1538 2 THEN
1460 1539 2 BEGIN
1461 1540 2 $exch_signal (exch$rt11_stblock);
1462 1541 2 sum_blocks = .seg [rt11hdr$w_start_block];
1463 1542 2 END;
```

```
: 1465      1543 3      | : Get a pointer to the first directory entry, then loop
: 1466      1544 3      | :
: 1467      1545 3      | ent = .seg + rt11hdr$k_length;
: 1468      1546 3      | end_segment_seen = false;
: 1469      1547 4      | WHILE (.ent - LSSU (.seg + rt11$k_dirseglen))
: 1470      1548 3      | DO
: 1471      1549 4      |     BEGIN
: 1472      1550 4      |         : Advance our count
: 1473      1551 4      |         :
: 1474      1552 4      |         :
: 1475      1553 4      |         CASE .ent [rt11ent$v_type] FROM 0 TO rt11ent$m_typ_end_segment OF
: 1476      1554 4      |         SET
: 1477      1555 4      |         [rt11ent$m_typ_tentative, rt11ent$m_typ_empty, rt11ent$m_typ_permanent] :
: 1478      1556 5      |             BEGIN
: 1479      1557 5      |                 $debug_print_fao ('entry type !XB, size !UL', .ent [rt11ent$b_type_byte], .ent [rt11ent$w_bl
: 1480      1558 5      |                 sum_blocks = .sum_blocks + .ent [rt11ent$w_blocks];
: 1481      1559 4      |             END;
: 1482      1560 4      |         [rt11ent$m_typ_end_segment] :
: 1483      1561 5      |             BEGIN
: 1484      1562 5      |                 end_segment_seen = true;
: 1485      1563 5      |                 EXITLOOP;
: 1486      1564 4      |             END;
: 1487      1565 4      |         [INRANGE, OVRANGE] :
: 1488      1566 5      |             BEGIN
: 1489      1567 5      |                 unknowns = true;
: 1490      1568 5      |                 $debug_print_fao ('(unknown) entry type !XB, size !UL', .ent [rt11ent$b_type_byte], .ent [rt
: 1491      1569 5      |                 sum_blocks = .sum_blocks + .ent [rt11ent$w_blocks];
: 1492      1570 4      |             END;
: 1493      1571 4      |         TES;
: 1494      1572 4      |         :
: 1495      1573 4      |         : Skip to the next entry
: 1496      1574 4      |         :
: 1497      1575 4      |         ent = .ent + rt11ent$k_length + .seg [rt11hdr$w_extra_bytes];
: 1498      1576 4      |         :
: 1499      1577 3      |         END;
: 1500      1578 3      |         :
: 1501      1579 3      |         : Holler if we didn't see the end segment
: 1502      1580 3      |         :
: 1503      1581 4      |         IF NOT (.end_segment_seen)
: 1504      1582 3      |         THEN
: 1505      1583 3      |             missing_end = true;
: 1506      1584 3      |         :
: 1507      1585 3      |         : Skip to the next segment
: 1508      1586 3      |         :
: 1509      1587 3      |         seg_num = .seg [rt11hdr$w_next_seg];
: 1510      1588 3      |         :
: 1511      1589 2      |     END;
```

```
1513 1590 2 ! Assume now that we have found a valid directory
1514 1591 2
1515 1592 2 status = true;
1516 1593 2
1517 1594 2 ! If we saw any bad entries, signal and write lock the volume
1518 1595 2
1519 1596 2 ! .unknowns OR .missing_end ! Unknown format of some entries or missing marker
1520 1597 2 THEN
1521 1598 2 BEGIN
1522 1599 2 LOCAL
1523 1600 2 sig_stat;
1524 1601 2
1525 1602 2 sig_stat = (IF .unknowns THEN exch$_rt11_unkent ELSE exch$_rt11_noend); ! Get the proper signal name
1526 1603 2
1527 1604 2 ! Signal if this is the first verify, or if we have write access
1528 1605 2
1529 1606 2 IF NOT .volb [volb$_v_verified]
1530 1607 2 OR
1531 1608 2 .volb [volb$_v_write]
1532 1609 2 THEN
1533 1610 2 $exch_signal (.sig_stat, 0, exch$_rt11_errlock);
1534 1611 2
1535 1612 2 ! Fail the directory validation this is not the first and we have write access
1536 1613 2
1537 1614 2 IF .volb [volb$_v_verified]
1538 1615 2 AND
1539 1616 2 .volb [volb$_v_write]
1540 1617 2 THEN
1541 1618 2 status = false;
1542 1619 2
1543 1620 2 volb [volb$_v_write] = false; ! Force a write lock
1544 1621 2 END;
1545 1622 2
1546 1623 2 ! Check if the block count is what we expected
1547 1624 2
1548 1625 2 $debug_print_fao ('volmaxblock !UL, sum_blocks !UL', .volb [volb$_l_volmaxblock], .sum_blocks);
1549 1626 2 IF .volb [volb$_l_volmaxblock] NEQ .sum_blocks
1550 1627 2 THEN
1551 1628 2 BEGIN
1552 1629 2
1553 1630 2 ! If the volume is larger than the device there is no hope
1554 1631 2
1555 1632 2 $debug_print_fao ('devmaxblock !UL, sum_blocks !UL', .volb [volb$_l_devmaxblock], .sum_blocks);
1556 1633 2 IF .sum_blocks GTRU .volb [volb$_l_devmaxblock]
1557 1634 2 THEN
1558 1635 2 BEGIN
1559 1636 2 $exch_signal (exch$_rt11_dirsize, 4, .volb [volb$_l_vol_ident_len], volb [volb$_t_vol_ident],
1560 1637 2 .sum_blocks, .volb [volb$_l_devmaxblock], exch$_rt11_errlock);
1561 1638 2 volb [volb$_v_write] = false;
1562 1639 2 END;
1563 1640 2
1564 1641 2 ! If this is the first time that the directory has been verified, store
1565 1642 2 the calculated value as the real value.
1566 1643 2
1567 1644 2 IF NOT .volb [volb$_v_verified]
1568 1645 2 THEN
1569 1646 2 BEGIN
```



```

: 1570      1647  4      $logic_check (3, (.volb [volb$l_volmaxblock] EQL .volb [volb$l_devmaxblock]), 190);
: 1571      1648  4      volb [volb$l_volmaxblock] = .sum_blocks;
: 1572      1649  4      END
: 1573      1650  4
: 1574      1651  4      ! Otherwise, we have corrupted the directory since we started
: 1575      1652  4      !
: 1576      1653  3      ELSE
: 1577      1654  3      $logic_check (0, (false), 316);
: 1578      1655  2      END;
: 1579      1656  2
: 1580      1657  2      ! Set the verification flag
: 1581      1658  2      !
: 1582      1659  2      volb [volb$v_verified] = true;
: 1583      1660  2
: 1584      1661  2      RETURN .status;
: 1585      1662  1      END;

```

```

.EXTRN EXCH$RT11_STBLOCK
.EXTRN EXCH$RT11_UNKENT
.EXTRN EXCH$RT11_NOEND
.EXTRN EXCH$RT11_ERRLOCK
.EXTRN EXCH$RT11_DIRSIZE

```

				OFFC 00000	.ENTRY	
			SE	04 C2 00002	SUBL2	#4, SP
			54	04 AC D0 00005	MOVL	VOLB, R4
			52	041B00F3 8F D0 00009	MOVL	#68878579, R2
			51	01D6 8F 3C 00010	MOVZWL	#470, R1
			50	54 D0 00015	MOVL	R4, R0
				00000000G EF 16 00018	JSB	EXCH\$UTIL_BLOCK_CHECK
				5A 7C 0001E	CLRQ	UNKNOWN
				59 D4 00020	CLRL	MISSING_END
			53	54 A4 D0 00022	MOVL	84(R4), RTV
			52	880E00F5 8F D0 00026	MOVL	#-2012348171, R2
			51	01D7 8F 3C 0002D	MOVZWL	#471, R1
			50	53 D0 00032	MOVL	RTV, R0
				00000000G EF 16 00035	JSB	EXCH\$UTIL_BLOCK_CHECK
			55	0C0E C3 9E 0003B	MOVAB	3086(R3), SEG
			58	01 D0 00040	MOVL	#1, SEG_NUM
			52	65 3C 00043	MOVZWL	(SEG), R2
			52	02 C4 00046	MULL2	#2, SUM_BLOCKS
			52	06 C0 00049	ADDL2	#6, SUM_BLOCKS
				58 D5 0004C 1\$:	TSTL	SEG_NUM
				03 12 0004E	BNEQ	2\$
				0084 31 00050	BRW	13\$
				00000000G EF 0110 8F BB 00053 2\$:	PUSHR	#^M<R4,R8>
			55	02 FB 00057	CALLS	#2, EXCH\$RT11_DIRSEG_GET
				50 D0 0005E	MOVL	R0, SEG
				03 12 00061	BNEQ	3\$
				0110 31 00063	BRW	24\$
52	08	A5	10	00 ED 00066 3\$:	CMPZV	#0, #16, 8(SEG), SUM_BLOCKS
				11 13 0006C	BEQL	4\$
				00000000G 8F DD 0006E	PUSHL	#EXCH\$RT11_STBLOCK
				01 FB 00074	CALLS	#1, LIB\$SIGNAL

			52	08	A5	3C	0007B		MOVZWL	8(SEG), SUM_BLOCKS	1541	
			53	0A	A5	9E	0007F	4\$:	MOVAB	10(R5), ENT	1545	
					56	D4	00083		CLRL	END_SEGMENT_SEEN	1546	
			50	0400	C5	9E	00085		MOVAB	1024(R5), R0	1547	
			57	06	A5	9E	0008A		MOVAB	6(SEG), R7	1575	
			50		53	D1	0008E	5\$:	CMP	ENT, R0	1547	
					37	1E	00091		BGEQU	11\$		
51	01	A3	04		00	EF	00093		EXTZV	#0, #4, 1(ENT), R1	1553	
		08	00		51	CF	00099		CASEL	R1, #0, #8		
0012		0015	0015		0012		0009D	6\$:	.WORD	7\$-6\$,-		
0012		0012	0012		0015		000A5			8\$-6\$,-		
					001E		000AD			8\$-6\$,-		
										7\$-6\$,-		
										8\$-6\$,-		
										7\$-6\$,-		
										8\$-6\$,-		
										7\$-6\$,-		
										7\$-6\$,-		
										7\$-6\$,-		
										9\$-6\$		
			5A		01	D0	000AF	7\$:	MOVL	#1, UNKNOWN	1567	
			6E	08	A3	3C	000B2	8\$:	MOVZWL	8(ENT), (SP)	1558	
			52		6E	C0	000B6		ADDL2	(SP), SUM_BLOCKS		
					05	11	000B9		BRB	10\$	1553	
			56		01	D0	000BB	9\$:	MOVL	#1, END_SEGMENT_SEEN	1562	
					0A	11	000BE		BRB	11\$	1561	
			51		67	3C	000C0	10\$:	MOVZWL	(R7), R1	1575	
			53	0E	A143	9E	000C3		MOVAB	14(R1)[ENT], ENT		
					C4	11	000C8		BRB	5\$	1547	
			03		56	E8	000CA	11\$:	BLBS	END_SEGMENT_SEEN, 12\$	1581	
			59		01	D0	000CD		MOVL	#1, MISSING_END	1583	
			58	02	A5	3C	000D0	12\$:	MOVZWL	2(SEG), SEG_NUM	1587	
					FF75	31	000D4		BRW	1\$	1523	
			5B		01	D0	000D7	13\$:	MOVL	#1, STATUS	1592	
			06		5A	E8	000DA		BLBS	UNKNOWN, 14\$	1596	
			3E		59	E9	000DD		BLBC	MISSING_END, 20\$		
			09		5A	E9	000E0		BLBC	UNKNOWN, 15\$	1602	
			50	00000000G	8F	D0	000E3	14\$:	MOVL	#EXCH\$RT11_UNKENT, SIG_STAT		
					07	11	000EA		BRB	16\$		
			50	00000000G	8F	D0	000EC	15\$:	MOVL	#EXCH\$RT11_NOEND, SIG_STAT		
				48	A4	95	000F3	16\$:	TSTB	72(R4)	1606	
					05	18	000F6		BGEQ	17\$		
	11	48	A4		05	E1	000F8		BBC	#5, 72(R4), 18\$	1608	
				00000000G	8F	DD	000FD	17\$:	PUSHL	#EXCH\$RT11_ERRLOCK	1610	
					7E	D4	00103		CLRL	-(SP)		
					50	DD	00105		PUSHL	SIG_STAT		
			00000000G	00	03	FB	00107		CALLS	#3, LIB\$SIGNAL		
					48	A4	95	0010E	18\$:	TSTB	72(R4)	1614
					07	18	00111		BGEQ	19\$		
	02	48	A4		05	E1	00113		BBC	#5, 72(R4), 19\$	1616	
					5B	D4	00118		CLRL	STATUS	1618	
			48		20	8A	0011A	19\$:	BICB2	#32, 72(R4)	1620	
			52	44	A4	D1	0011E	20\$:	CMP	68(R4), SUM_BLOCKS	1626	
					49	13	00122		BEQL	23\$		
			40	A4	52	D1	00124		CMP	SUM_BLOCKS, 64(R4)	1633	
					24	1B	00128		BLEQU	21\$		
				00000000G	8F	DD	0012A		PUSHL	#EXCH\$RT11_ERRLOCK	1637	
				40	A4	DD	00130		PUSHL	64(R4)		
					52	DD	00133		PUSHL	SUM_BLOCKS		

		69	A4	9F	00135	PUSHAB	105(R4)	
		65	A4	DD	00138	PUSHL	101(R4)	
			04	DD	0013B	PUSHL	#4	
		00000000G	00	8F	DD	0013D	PUSHL	#EXCH\$ RT11 DIRSIZE
		48	A4	07	FB	00143	CALLS	#7, LIB\$SIGNAL
			48	20	8A	0014A	BICB2	#32, 72(R4)
				A4	95	0014E	TSTB	72(R4)
				06	19	00151	BLSS	22\$
		44	A4	52	D0	00153	MOVL	SUM_BLOCKS, 68(R4)
				14	11	00157	BRB	23\$
			7E	8F	3C	00159	MOVZWL	#316, -(SP)
				01	DD	0015E	PUSHL	#1
		00000000G	00	8F	DD	00160	PUSHL	#EXCH\$ BADLOGIC
		48	A4	03	FB	00166	CALLS	#3, LIB\$STOP
			80	8F	88	0016D	BISB2	#128, 72(R4)
			50	5B	D0	00172	MOVL	STATUS, R0
					04	00175	RET	
			50	D4	00176	CLRL	R0	
				04	00178	RET		

.....  
1638  
1644  
.....  
1648  
1644  
1654  
.....  
1659  
1661  
.....  
1662  
.....

; Routine Size: 377 bytes, Routine Base: EXCH\$RTACP\_CODE + 07F9

EXCH\$RTACP  
V04-000

RT11 directory routines  
exch\$rtacp\_verify\_directory (volb)

1 7  
16-Sep-1984 01:19:05  
14-Sep-1984 12:29:08

VAX-11 Bliss-32 V4.0-742  
[EXCHNG.SRC]EXCRTACP.B32;1

Page 50  
(21)

EXC  
V04

: 1587 1663 1 END  
: 1588 1664 0 ELUDOM

.EXTRN LIB\$SIGNAL, LIB\$STOP

PSECT SUMMARY

Name	Bytes	Attributes
EXCH\$RTACP_CODE	2418	NOVEC,NOWRT, RD, EXE,NOSHR, LCL, REL, CON,NOPIC,ALIGN(2)

Library Statistics

File	----- Total	Symbols Loaded	----- Percent	Pages Mapped	Processing Time
\$255\$DUA28:[SYSLIB]LIB.L32;1	18619	1	0	1000	00:01.9
_\$255\$DUA28:[EXCHNG.OBJ]EXCLIB.L32;1	1151	74	6	79	00:01.4

COMMAND QUALIFIERS

BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS\$:EXCRTACP/OBJ=OBJ\$:EXCRTACP MSRCS\$:EXCRTACP/UPDATE=(ENHS\$:EXCRTACP)

: Size: 2418 code + 0 data bytes  
: Run Time: 00:51.6  
: Elapsed Time: 02:35.2  
: Lines/CPU Min: 1936  
: Lexemes/CPU-Min: 17630  
: Memory Used: 236 pages  
: Compilation Complete



