

EEEEEEEEEEEEEEEE XXX XXX CCCCCCCCCCCC HHH HHH NNN NNN GGGGGGGGGGG
EEEEEEEEEEEEEEEE XXX XXX CCCCCCCCCCCC HHH HHH NNN NNN GGGGGGGGGGG
EEEEEEEEEEEEEEEE XXX XXX CCCCCCCCCCCC HHH HHH NNN NNN GGGGGGGGGGG
EEE XXX XXX CCC CCCCCCCCCCCC HHH HHH NNN NNN GGG
EEE XXX XXX CCC CCCCCCCCCCCC HHH HHH NNN NNN GGG
EEE XXX XXX CCC CCCCCCCCCCCC HHH HHH NNN NNN GGG
EEE XXX XXX CCC CCCCCCCCCCCC HHH HHH NNN NNN GGG
EEE XXX XXX CCC CCCCCCCCCCCC HHH HHH NNN NNN GGG
EEE XXX XXX CCC CCCCCCCCCCCC HHH HHH NNN NNN GGG
EEE XXX XXX CCC CCCCCCCCCCCC HHH HHH NNN NNN GGG
EEE XXX XXX CCC CCCCCCCCCCCC HHH HHH NNN NNN GGG
EEEEEEEEEEEEEEEE XXX XXX CCCCCCCCCCCC HHH HHH NNN NNN GGGGGGGGG
EEEEEEEEEEEEEEEE XXX XXX CCCCCCCCCCCC HHH HHH NNN NNN GGGGGGGGG
EEEEEEEEEEEEEEEE XXX XXX CCCCCCCCCCCC HHH HHH NNN NNN GGGGGGGGG

I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
[
\
]
^
_
`
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x
y
z
{|

```

EEEEEEEEEE XX XX CCCCCCCC PPPPPPPP DDDDDDDD PPPPPPPP
EEEEEEEEEE XX XX CCCCCCCC PPPPPPPP DDDDDDDD PPPPPPPP
EE XX XX CC PP PP DD DD PP PP
EE XX XX CC PP PP DD DD PP PP
EE XX XX CC PP PP DD DD PP PP
EEEEEEEE XX XX CC PPPPPPPP DD DD PPPPPPPP
EEEEEEEE XX XX CC PPPPPPPP DD DD PPPPPPPP
EE XX XX CC PP PP DD DD PP
EE XX XX CC PP PP DD DD PP
EEEEEEEEEE XX XX CCCCCCCC PP DDDDDDDD PP
EEEEEEEEEE XX XX CCCCCCCC PP DDDDDDDD PP

```

```

LL IIIIII SSSSSSSS
LL IIIIII SSSSSSSS
LL II SS
LL II SS
LL II SS
LL II SSSSSS
LL II SSSSSS
LL II SS
LL II SS
LL II SS
LLLLLLLLLL IIIIII SSSSSSSS
LLLLLLLLLL IIIIII SSSSSSSS

```

```

1 0001 0 MODULE  exch$pdp                                %TITLE 'Small PDP-11 record structure routines'
2 0002 0
3 0003 0          (
4 0004 0          IDENT = 'V04-000'
5 0005 0          ADDRESSING_MODE (EXTERNAL=LONG_RELATIVE, NONEXTERNAL=WORD_RELATIVE)
6 0006 0          ) =
7 0007 1 BEGIN
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:      Specific routines for record structures from small PDP-11
35 0035 1                systems, RT-11 and DOS-11
36 0036 1
37 0037 1 ENVIRONMENT:   VAX/VMS User mode
38 0038 1
39 0039 1 AUTHOR:        CW Hobbs                CREATION DATE: 26-Aug-1982
40 0040 1                                         Split from RT11 module: 28-Nov-1982
41 0041 1
42 0042 1 MODIFIED BY:
43 0043 1
44 0044 1
45 0045 1 --
46 0046 1
47 0047 1 Include files:
48 0048 1
49 0049 1 MACRO $module_name_string = 'exch$pdp' %;          ! The require file needs to know our module name
50 0050 1 REQUIRE 'SRCS:EXCREQ'                            ! Facility-wide require file
51 0051 1

```

```

53 0148 1 %SBTTL 'Module table of contents'
54 0149 1
55 0150 1 ! Module table of contents:
56 0151 1 !
57 0152 1 FORWARD ROUTINE
58 0153 1     pdp_buffer_advance_read,      ! Read some more data into the ctx buffer
59 0154 1     pdp_buffer_advance_write,  ! Write some data from the ctx buffer
60 0155 1     pdp_buffer_check : jsb_r2r3, ! Check the buffer
61 0156 1     pdp_buffer_update : jsb_r2r3, ! Update the buffer pointers in the context block
62 0157 1     pdp_check_ctx : NOVALUE,     ! Check the context block for consistency
63 0158 1     pdp_copy_binary_record : NOVALUE, ! Copy a formatted-binary record
64 0159 1     pdp_copy_stream_record,    ! Copy a record to a stream format record
65 0160 1     exch$pdp_filter_filename,    ! Remove invalid characters from a filename
66 0161 1     pdp_find_binary_record,    ! Find a formatted binary record in a given buffer
67 0162 1     pdp_find_stream_record,    ! Find a stream record in a given buffer
68 0163 1     exch$pdp_flush_write_buffer, ! Flush any records waiting in the output buffer
69 0164 1     exch$pdp_get,               ! Get routine dispatch
70 0165 1     pdp_get_binary : jsb_get,   ! Get formatted binary record
71 0166 1     pdp_get_fixed : jsb_get,    ! Get fixed-length record
72 0167 1     pdp_get_stream : jsb_get,   ! Get stream format record
73 0168 1     exch$pdp_put,               ! Put dispatcher
74 0169 1     pdp_put_binary : jsb_put,   ! Put formatted binary record
75 0170 1     pdp_put_fixed : jsb_put,    ! Put fixed-length record
76 0171 1     pdp_put_stream : jsb_put,  ! Put stream format record
77 0172 1     ;
78 0173 1
79 0174 1 ! EXCHANGE facility routines
80 0175 1 !
81 0176 1 EXTERNAL ROUTINE
82 0177 1     exch$io_dos11_read,           ! Read blocks from a sequential device
83 0178 1     exch$io_dos11_skip_record,  ! Space over blocks on a sequential device
84 0179 1     exch$io_dos11_write,      ! Write blocks to a sequential device
85 0180 1     exch$io_rt11_read,        ! Read blocks from a random access device
86 0181 1     exch$io_rt11_write,       ! Write blocks to a random access device
87 0182 1     exch$rt11_bad_file : NOVALUE, ! Erase an RT11 file because of error
88 0183 1     exch$util_vm_allocate      ! Get some virtual memory
89 0184 1     ;
90 0185 1
91 0186 1 ! Equated symbols:
92 0187 1 !
93 0188 1 ! LITERAL
94 0189 1 !
95 0190 1 !
96 0191 1 ! Bound declarations:
97 0192 1 !
98 0193 1 ! BIND
99 0194 1 !
100 0195 1 !
101 0196 1 ! Local macros
102 0197 1 !
103 0198 1 MACRO
104 M 0199 1     $$show_context = $trace_print_fao ('cur !SL, byt !SL, eof !SL, base !SL, high !SL, wr !SL',
105 M 0200 1     .ctx [ctx$_cur_block], .ctx [ctx$_cur_byte], .ctx [ctx$_eof_block],
106 M 0201 1     .ctx [ctx$_buf_base_block], .ctx [ctx$_buf_high_block], .ctx [ctx$_high_block_wri
107 0202 1     %;

```

```
109 0203 1 GLOBAL ROUTINE pdp_buffer_advance_read (ctx : $ref_bblock) = %SBTTL 'pdp_buffer_advance_read (ctx)'  
110 0204 2 BEGIN  
111 0205 2 ++  
112 0206 2  
113 0207 2 FUNCTIONAL DESCRIPTION:  
114 0208 2  
115 0209 2 Move the current block to the leftmost position in the buffer, and read in new blocks  
116 0210 2  
117 0211 2 INPUTS:  
118 0212 2  
119 0213 2 ctx - ctx pointer to context for an open RT11 file  
120 0214 2  
121 0215 2 IMPLICIT INPUTS:  
122 0216 2  
123 0217 2 none  
124 0218 2  
125 0219 2 OUTPUTS:  
126 0220 2  
127 0221 2 none  
128 0222 2  
129 0223 2 IMPLICIT OUTPUTS:  
130 0224 2  
131 0225 2 none  
132 0226 2  
133 0227 2 ROUTINE VALUE:  
134 0228 2  
135 0229 2 true if success, false if any error  
136 0230 2  
137 0231 2 SIDE EFFECTS:  
138 0232 2  
139 0233 2 error conditions will be signaled  
140 0234 2 --  
141 0235 2  
142 0236 2 $dbgtrc_prefix ('pdp_buffer_advance_read> ');  
143 0237 2  
144 0238 2 LOCAL  
145 0239 2 blks_in_use,  
146 0240 2 blks_to_read,  
147 0241 2 buf_start, ! Pointer to next byte in the buffer  
148 0242 2 buf_end, ! -> one past the end of buffer  
149 0243 2 buf_len, ! Length of good part of buffer  
150 0244 2 stafus  
151 0245 2 ;  
152 0246 2  
153 0247 2 BIND  
154 0248 2 base = ctx [ctx$l_buf_base_block],  
155 0249 2 buf = ctx [ctx$a_buffer],  
156 0250 2 byt = ctx [ctx$l_cur_byte],  
157 0251 2 cur = ctx [ctx$l_cur_block],  
158 0252 2 eof = ctx [ctx$l_eof_block],  
159 0253 2 high = ctx [ctx$l_buf_high_block],  
160 0254 2 filb = ctx [ctx$a_assoc_filb] : $ref_bblock,  
161 0255 2 volb = ctx [ctx$a_assoc_volb] : $ref_bblock  
162 0256 2 ;  
163 0257 2  
164 0258 2  
165 0259 2 $trace_print_lit ('entry');
```

```
166 0260 2
167 0261 2 $check_call (2, pdp_check_ctx, .ctx, 441);          ! $block_check (2, .ctx, (dos11ctx or rt11ctx), 441)
168 0262 2
169 0263 2 ! If the current block is at the beginning or the high block is EOF, we have made a grievous error
170 0264 2
171 0265 2 $$show_context;
172 0266 2 $logic_check (3, ((.cur GEQU .base) AND ((.cur LEQU .high+1) OR (.high EQL .base-1))), 214);
173 0267 2
174 0268 2 ! Get a pointer to the place to start shuffling, and a pointer to the first byte past the end of the buffer
175 0269 2
176 0270 2 $logic_check (2, (.buf NEQ 0), 181);
177 0271 2 buf_start = .buf + ((.cur - .base) * 512);
178 0272 2 buf_end = .buf + ((1 + .high - .base) * 512);
179 0273 2 buf_len = .buf_end - .buf_start;
180 0274 2 $logic_check (2, (.buf_len [SSU 65536]), 116); ! Short-sighted architects
181 0275 2
182 0276 2 ! If current block is the base block, do some more looking.
183 0277 2
184 0278 2 IF (.cur EQL .base AND .high NEQ .base-1) ! initial condition
185 0279 2 THEN
186 0280 2 BEGIN
187 P 0281 2 $trace_print_fao ('*cur is base* buf_start !XL, buf !XL, buf_len !XW, ctx$k_buffer_length !XW',
188 0282 2 .buf_start, .buf, .buf_len, ctx$k_buffer_length);
189 0283 2 $logic_check (3, ((.buf_start EQL .buf) AND (.buf_len EQL ctx$k_buffer_length)), 215);
190 0284 2
191 0285 2 ! If there are non-null characters in the end of the buffer, then the record is too big and we have an e
192 0286 2
193 0287 2 IF CH$NEQ (0, .cur, .buf_len - .byt, .buf + .byt, 0)
194 0288 2 THEN
195 0289 2 $exch_signal_return (exch$_rectoobig, 2, .filb [filb$_result_name_len], filb [filb$_result_name])
196 0290 2
197 0291 2 ! OK, we have some data in the first block, and nulls to the end of the buffer. We will slide over the
198 0292 2 and refresh the end of the buffer, since stream and binary formats skip nulls. This is done so that w
199 0293 2 handle a stream file with a large number of zeroed blocks at the end.
200 0294 2
201 0295 2 ELSE
202 0296 2 BEGIN
203 0297 2 $trace_print_lit ('*slide one block*');
204 0298 2 cur = .high; ! "Slide" it to the end
205 0299 2 buf_len = 512; ! One good block
206 0300 2 END;
207 0301 2 END
208 0302 2
209 0303 2 ELSE
210 0304 2 BEGIN
211 0305 2
212 0306 2 ! Current not the base, move the good data to the start of the buffer
213 0307 2
214 0308 2 $trace_print_fao ('*cur not base* buf_start !XL, buf !XL, buf_len !XW', .buf_start, .buf, .buf_len);
215 0309 2 IF .buf_start NEQ .buf
216 0310 2 THEN
217 0311 2 BEGIN
218 0312 2 $trace_print_lit ('shuffling data to the start of the buffer');
219 0313 2 CH$MOVE (.buf_len, .buf_start, .buf);
220 0314 2 END;
221 0315 2 END;
222 0316 2
```

```
223 0317 2 ! Change the base pointer to show what we just did, buf_high_block is still valid
224 0318 2
225 0319 2 base = .cur;
226 0320 2
227 0321 2 ! Read a chunk into the buffer
228 0322 2
229 0323 2 blks_in_use = .buf_len / 512;
230 0324 2 blks_to_read = ctx$buffer_blocks - .blks_in_use; ! Blocks left in buffer
231 0325 2 IF (.eof - .high) GTR 0 ! Blocks left in file
232 0326 2 THEN
233 0327 2     blks_to_read = MINU (.blks_to_read, (.eof - .high));
234 0328 2
235 0329 2 ! If all of the blocks are in use, then we have no room to fit more data into the buffer. Return with a rec
236 0330 2 error, which our caller can examine.
237 0331 2
238 0332 2 IF .blks_in_use GEQU ctx$buffer_blocks
239 0333 2 THEN
240 0334 2     RETURN exch$_stmrecfmt;
241 0335 2
242 P 0336 2 $trace_print_fao ('blocks in use !UL, blocks to read !UL, ctx$buffer_blocks !UL',
243 0337 2     .blks_in_use, .blks_to_read, ctx$buffer_blocks);
244 0338 2 $$show_context;
245 0339 2 $logic_check (2, (.blks_to_read GTRU 0), 118);
246 0340 2
247 0341 2 ! Perform the appropriate read operation depending on the volume type
248 0342 2
249 0343 2 IF .volb [volb$_vol_format] EQL volb$_vfmt_rt11
250 0344 2 THEN
251 0345 2     BEGIN
252 0346 2     IF NOT (status = exch$_io_rt11_read (.volb, ! All the rms stuff hangs from here
253 0347 2     .high + 1, ! First block to read
254 0348 2     .blks_to_read, ! Number of blocks
255 0349 2     .buf + .buf_len)) ! Address of the I/O buffer
256 0350 2     THEN
257 0351 2     RETURN .status;
258 0352 2     END
259 0353 2 ELSE
260 0354 2     BEGIN
261 0355 2     LOCAL
262 0356 2     bp, ! Buffer pointer
263 0357 2     bc; ! Block count
264 0358 2
265 0359 2     bc = .blks_to_read; ! Number of blocks to read
266 0360 2     bp = .buf + .buf_len; ! Address to put first block
267 0361 2
268 0362 2     WHILE 1
269 0363 2     DO
270 0364 2     BEGIN
271 0365 2     ! Read from the tape
272 0366 2     !
273 0367 2     !
274 0368 2     status = exch$_io_dos11_read ( .volb, ! All the stuff hangs from here
275 0369 2     .bp); ! Address of the I/O buffer
276 0370 2     !
277 0371 2     ! If the read didn't work, do some checking
278 0372 2     !
279 0373 2     IF NOT .status
```

```

280 0374 4 THEN
281 0375 5 BEGIN
282 0376 5 IF .status EQL ss$_endoffile
283 0377 5 OR
284 0378 5 .status EQL ss$_endoftape
285 0379 5 THEN
286 0380 6 BEGIN
287 0381 6 strace_print_lit ('registered END-C'-FILE');
288 0382 6 $$show_context;
289 0383 6 eof = MAX (0, (.high + (.blks_to_read - .bc))); ! Set the eof block to zero or more
290 0384 6 blks_to_read = .blks_to_read - .bc; ! Adjust so that high block gets set right;
291 0385 6 EXITLOOP;
292 0386 6 END
293 0387 5 ELSE
294 0388 5 RETURN .status; ! Return the error status
295 0389 4 END;
296 0390 4
297 0391 4 ! Adjust our pointers
298 0392 4
299 0393 4 bp = .bp + 512; ! Move to the next block
300 0394 4 bc = .bc - 1;
301 0395 4 IF .bc LEQ 0 THEN EXITLOOP; ! Exit if all have been read
302 0396 4
303 0397 3 END;
304 0398 2 END;
305 0399 2
306 0400 2 ! Change the high block pointer to show what we just did
307 0401 2
308 0402 2 high = .high + .blks_to_read;
309 0403 2
310 0404 2 RETURN true;
311 0405 2
312 0406 1 END;

```

```

.TITLE EXCHSPDP Small PDP-11 record structure routines
.IDENT \V04-000\

```

```

.EXTRN EXCH$IO_DOS11_READ
.EXTRN EXCH$IO_DOS11_SKIP_RECORD
.EXTRN EXCH$IO_DOS11_WRITE
.EXTRN EXCH$IO_RT11_READ
.EXTRN EXCH$IO_RT11_WRITE
.EXTRN EXCH$RTT1_BAD_FILE
.EXTRN EXCH$UTIL_VM_ALLOCATE
.EXTRN PDP_CHECK_CTX, EXCH$_BADLOGIC
.EXTRN EXCH$_RECTOOBIG
.EXTRN EXCH$_STMRECFMT

```

```

.PSECT EXCHSPDP_CODE, NOWRT, 2

```

OFFC 00000

```

.ENTRY PDP_BUFFER_ADVANCE_READ, Save R2,R3,R4,R5,- : 0203
R6,R7,R8,R9,R10,R11
MOVAB LIB$STOP, R11
MOVL #EXCH$_BADLOGIC, R10
MOVL CTX, R7 : 0248
MOVZWL #441, -(SP) : 0261

```

```

5B 00000000G 00 9E 00002
5A 00000000G 8F D0 00009
57 04 AC D0 00010
7E 01B9 8F 3C 00014

```


				57	DD	00019		PUSHL	R7			
	00000000G	00		02	FB	0001B		CALLS	#2, PDP_CHECK_CTX			
		58	18	A7	DO	00022		MOVL	24(R7), -R8			0270
				0B	12	00026		BNEQ	1\$			
		7E	B5	8F	9A	00028		MOVZBL	#181, -(SP)			
				01	DD	0002C		PUSHL	#1			
				5A	DD	0002E		PUSHL	R10			
		6B		03	FB	00030		CALLS	#3, LIB\$STOP			
53	1C	A7	2C	A7	C3	00033	1\$:	SUBL3	44(R7), 28(R7), R3			0271
53		53		09	78	00039		ASHL	#9, R3, R3			
52		53		58	C1	0003D		ADDL3	R8, R3, BUF_START			
		59	30	A7	DO	00041		MOVL	48(R7), R9			0272
50		59	2C	A7	C3	00045		SUBL3	44(R7), R9, R0			
50		50		09	78	0004A		ASHL	#9, R0, R0			
		50	0200	C048	9E	0004E		MOVAB	512(R0)[R8], BUF_END			
56		50		52	C3	00054		SUBL3	BUF_START, BUF_END, BUF_LEN			0273
	00010000	8F		56	D1	00058		CMPL	BUF_LEN, #65536			0274
				0B	1F	0005F		BLSSU	2\$			
		7E	74	8F	9A	00061		MOVZBL	#116, -(SP)			
				01	DD	00065		PUSHL	#1			
				5A	DD	00067		PUSHL	R10			
		6B		03	FB	00069		CALLS	#3, LIB\$STOP			
		2C	A7	1C	A7	D1	2\$:	CMPL	28(R7), 44(R7)			0278
				45	12	00071		BNEQ	4\$			
50		2C	A7	01	C3	00073		SUBL3	#1, 44(R7), R0			
		50		59	D1	00078		CMPL	R9, R0			
				3B	13	0007B		BEQL	4\$			
50		56	24	A7	C3	0007D		SUBL3	36(R7), BUF_LEN, R0			0287
50	00	1C	B7	00	2D	00082		CMPC5	#0, @28(R7), #0, R0, @36(R7)[R8]			
			24	B748		00088						
				20	13	0008B		BEQL	3\$			
		52	00000000G	8F	DO	0008D		MOVL	#EXCH\$_RECTOOBIG, TEMP			0289
		50		10	A7	DO		MOVL	16(R7), R0			
				5A	A0	9F		PUSHAB	90(R0)			
				3A	A0	DD		PUSHL	58(R0)			
				02	DD	0009E		PUSHL	#2			
				52	DD	000A0		PUSHL	TEMP			
	00000000G	00		04	FB	000A2		CALLS	#4, LIB\$SIGNAL			
		50		52	DO	000A9		MOVL	TEMP, R0			
				04	04	000AC		RET				
		1C	A7	59	DO	000AD	3\$:	MOVL	R9, 28(R7)			0298
		56	0200	8F	3C	000B1		MOVZWL	#512, BUF_LEN			0299
				09	11	000B6		BRB	5\$			0278
		58		52	D1	000B8	4\$:	CMPL	BUF_START, R8			0309
				04	13	000BB		BEQL	5\$			
68		62		56	28	000BD		MOV3	BUF_LEN, (BUF_START), (R8)			0313
		2C	A7	1C	A7	DO	5\$:	MOVL	28(R7), 44(R7)			0319
50		56	00000200	8F	C7	000C6		DIVL3	#512, BUF_LEN, BLKS_IN_USE			0323
52		0C		50	C3	000CE		SUBL3	BLKS_IN_USE, #12, BLKS_TO_READ			0324
		59	20	A7	D1	000D2		CMPL	32(R7), -R9			0325
				13	15	000D6		BLEQ	7\$			
53		20	A7	59	C3	000D8		SUBL3	R9, 32(R7), R3			0327
			51	52	DO	000DD		MOVL	BLKS_TO_READ, R1			
			53	51	D1	000E0		CMPL	R1, R3			
				03	1B	000E3		BLEQU	6\$			
		51		53	DO	000E5		MOVL	R3, R1			
		52		51	DO	000E8	6\$:	MOVL	R1, BLKS_TO_READ			

	0C		50	D1	000EB	7\$:	CMPL	BLKS_IN_USE, #12	:	0332	
			08	1F	000EE		BLSSU	8\$:		
	50	00000000G	8F	D0	000F0		MOVL	#EXCHS_STMRECFMT, R0	:	0334	
				04	000F7		RET		:		
				52	D5	000F8	8\$:	TSTL	BLKS_TO_READ	:	0339
				0B	12	000FA		BNEQ	9\$:	
	7E	76	8F	9A	00JFC		MOVZBL	#118, -(SP)	:		
				01	DD	00100		PUSHL	#1	:	
				5A	DD	00102		PUSHL	R10	:	
	6B			03	FB	00104		CALLS	#3, LIB\$STOP	:	
	53	14	A7	D0	00107	9\$:	MOVL	20(R7), R3	:	0343	
	0-		58	A3	91	0010B		CMPB	88(R3), #3	:	
				15	12	0010F		BNEQ	10\$:	
				6648	9F	00111		PUSHAB	(BUF_LEN)[R8]	:	0349
				52	DD	00114		PUSHL	BLKS_TO_READ	:	0348
				01	A9	9F	00116	PUSHAB	1(R9)	:	0347
				53	DD	00119		PUSHL	R3	:	0346
	00000000G		EF	04	FB	0011B		CALLS	#4, EXCH\$IO_RT11_READ	:	
			42	50	EB	00122		BLBS	STATUS, 15\$:	
				04	00125		RET		:	0351	
				52	D0	00126	10\$:	MOVL	BLKS_TO_READ, BC	:	0359
	55		58	56	C1	00129		ADDL3	BUF_LEN, R8, BP	:	0360
				28	BB	0012D	11\$:	PUSHR	#MZR3, R5	:	0368
	00000000G		EF	02	FB	0012F		CALLS	#2, EXCH\$IO_DOS11_READ	:	
			26	50	EB	00136		BLBS	STATUS, 14\$:	0373
	00000870		8F	50	D1	00139		CMPL	STATUS, #2160	:	0376
				09	13	00140		BEQL	12\$:	
	00000878		8F	50	D1	00142		CMPL	STATUS, #2168	:	0378
				23	12	00149		BNEQ	16\$:	
	51		52	54	C3	0014B	12\$:	SUBL3	BC, BLKS_TO_READ, R1	:	0383
			51	59	C0	0014F		ADDL2	R9, R1	:	
				02	18	00152		BGEQ	13\$:	
				51	D4	00154		CLRL	R1	:	
	20		A7	51	D0	00156	13\$:	MOVL	R1, 32(R7)	:	
			52	54	C2	0015A		SUBL2	BC, BLKS_TO_READ	:	0384
				08	11	0015D		BRB	15\$:	0380
			55	C5	9E	0015F	14\$:	MOVAB	512(R5), BP	:	0393
			C6	54	F5	00164		SOBGTR	BC, 11\$:	0394
	30		A7	52	C0	00167	15\$:	ADDL2	BLKS_TO_READ, 48(R7)	:	0402
			50	01	D0	0016B		MOVL	#1, R0	:	0404
				04	0016E	16\$:	RET		:	0406	

; Routine Size: 367 bytes, Routine Base: EXCHSPDP_CODE + 0000

```
314 0407 1 GLOBAL ROUTINE pdp_buffer_advance_write (ctx : $ref_bblock) = %SBTTL 'pdp_buffer_advance_write (ctx)'  
315 0408 BEGIN  
316 0409 ++  
317 0410  
318 0411 FUNCTIONAL DESCRIPTION:  
319 0412  
320 0413 Write the complete blocks in the buffer, then move the current block to the leftmost position in the  
321 0414 buffer.  
322 0415  
323 0416 INPUTS:  
324 0417  
325 0418 ctx - ctx pointer to context for an open RT11 file  
326 0419  
327 0420 IMPLICIT INPUTS:  
328 0421  
329 0422 none  
330 0423  
331 0424 OUTPUTS:  
332 0425  
333 0426 none  
334 0427  
335 0428 IMPLICIT OUTPUTS:  
336 0429  
337 0430 none  
338 0431  
339 0432 ROUTINE VALUE:  
340 0433  
341 0434 true if success, false if any error  
342 0435  
343 0436 SIDE EFFECTS:  
344 0437  
345 0438 error conditions will be signaled  
346 0439 --  
347 0440  
348 0441 $dbgtrc_prefix ('pdp_buffer_advance_write >');  
349 0442  
350 0443 LOCAL  
351 0444 temp,  
352 0445 blks_to_write,  
353 0446 buf_start, ! Pointer to next byte in the buffer  
354 0447 buf_end, ! -> one past the end of buffer  
355 0448 buf_len, ! Length of good part of buffer  
356 0449 status  
357 0450 ;  
358 0451  
359 0452 BIND  
360 0453 base = ctx [ctx$l_buf_base_block],  
361 0454 buf = ctx [ctx$a_buffer],  
362 0455 cur = ctx [ctx$l_cur_block],  
363 0456 eof = ctx [ctx$l_eof_block],  
364 0457 high = ctx [ctx$l_buf_high_block],  
365 0458 filb = ctx [ctx$a_assoc_filb] : $ref_bblock,  
366 0459 volb = ctx [ctx$a_assoc_volb] : $ref_bblock  
367 0460 ;  
368 0461  
369 0462  
370 0463 $trace_print_lit ('entry');
```

```
371 0464 2
372 0465 2 $check_call (2, pdp_check_ctx, .ctx, 458);      ! $block_check (2, .ctx, (), 458);
373 0466 2
374 0467 2 ! If the current block is before the beginning or the high block past EOF, we have made a grievous error
375 0468 2
376 0469 2 $logic_check (2, ((.cur GEQU .base) AND (.high LEQU .eof)), 242);
377 0470 2
378 0471 2 ! How many full blocks do we have?
379 0472 2
380 0473 2 blks_to_write = .cur - .base;
381 0474 2
382 0475 2 ! Get a pointer to the first partial block, the end of the buffer, and the length from the first partial to
383 0476 2 the end of the block
384 0477 2
385 0478 2 $logic_check (2, (.buf NEQ 0), 194);
386 0479 2 buf_start = .buf + ((.cur - .base) * 512);
387 0480 2 buf_end = .buf + ((1 + .high - .base) * 512);
388 0481 2 buf_len = .buf_end - .buf_start;
389 0482 2 $logic_check (2, (.buf_len [SSU 65536]), 173);      ' Short-sighted architects
390 0483 2
391 0484 2 ! Do a flush operation if necessary. The final partial block will be padded with nulls.
392 0485 2
393 0486 2 IF ((.ctx [ctx$v_flush])      ! Has a flush been requested
394 0487 2 AND
395 0488 2 (.ctx [ctx$l_cur_byte] NEQ 0))      ! And is there a partial block waiting
396 0489 2 THEN
397 0490 2 BEGIN
398 0491 2
399 0492 2 blks_to_write = .blks_to_write + 1;      ! Adjust the block count for the partial
400 0493 2
401 0494 2 CHSFILL (0, .buf_len - .ctx [ctx$l_cur_byte], .buf_start + .ctx [ctx$l_cur_byte]);
402 0495 2 END;
403 0496 2
404 0497 2 ! If we are flushing, set the eof block so that we may update the entry when we close (DOS-11 only)
405 0498 2
406 0499 2 IF (.ctx [ctx$v_flush])
407 0500 2 AND
408 0501 2 (.eof EQL -1)      ! DOS-11 has -1 for an EOF block
409 0502 2 THEN
410 0503 2 BEGIN
411 0504 2 $trace_print_lit ('flushing...');
412 0505 2 $$show_context;
413 0506 2 eof = .base + .blks_to_write - 1;
414 0507 2 END;
415 0508 2
416 P 0509 2 $trace_print_fao ('buf !XL, buf_start !XL, buf_end !XL, buf_len !XL, blocks to write !UL',
417 0510 2 .buf, .buf_start, .buf_end, .buf_len, .blks_to_write);
418 0511 2 $$show_context;
419 0512 2
420 0513 2 ! If no blocks, we don't have any more to do
421 0514 2
422 0515 2 IF .blks_to_write EQL 0
423 0516 2 THEN
424 0517 2 RETURN true;
425 0518 2
426 0519 2 ! Write the front chunk from the buffer, operation depends on the volume type
427 0520 2
```

```

428 0521 2 $logic check (2, ((.blks_to_write GTRU 0) AND (.blks_to_write LEQU ctx$sk_buffer_blocks)), 174);
429 0522 2 IF .volb [volb$b_vol_format] EQL volb$sk_vfmt_rtl1
430 0523 2 THEN
431 0524 2 BEGIN
432 0525 2
433 0526 4 IF NOT (status = exch$io_rtl1_write ( .volb,           ! All the rms stuff hangs from here
434 0527 4 .base,           ! First block to write
435 0528 4 .blks_to_write, ! Number of blocks
436 0529 4 .buf))           ! Address of the I/O buffer
437 0530 2 THEN
438 0531 2 BEGIN
439 0532 2 exch$rtl1_bad_file (.filb);
440 0533 2 RETURN .status;
441 0534 2 END;
442 0535 2 END
443 0536 2 ELSE
444 0537 2 BEGIN
445 0538 2 LOCAL
446 0539 2 bl,           ! Buffer length
447 0540 2 bp,           ! Buffer pointer
448 0541 2 bc;         ! Block count
449 0542 2
450 0543 2 bl = 512;       ! Most blocks are 512 bytes
451 0544 2 bc = .blks_to_write; ! Number of blocks to write
452 0545 2 bp = .buf;    ! Address to find first block
453 0546 2
454 0547 2 WHILE 1
455 0548 2 DO
456 0549 2 BEGIN
457 0550 2
458 0551 2 ! See if we are writing a final, short block
459 0552 2
460 0553 2 IF .ctx [ctx$v_flush]           ! Only if we are flushing
461 0554 2 THEN
462 0555 2 IF .bc EQL 1                 ! And if we are writing the last block
463 0556 2 THEN
464 0557 2 IF .ctx [ctx$l_cur_byte] NEQ 0 ! And if the block is partial
465 0558 2 THEN
466 0559 2 bl = .ctx [ctx$l_cur_byte]; ! Then the length is that partial
467 0560 2
468 0561 2 ! Write to the tape
469 0562 2
470 0563 2 status = exch$io_dos11_write ( .volb, ! All the stuff hangs from here
471 0564 2 .bp,           ! Address of the I/O buffer
472 0565 2 .bl);       ! Length of the I/O buffer
473 0566 2
474 0567 2 ! If the write didn't work, mark the buffer as empty before returning
475 0568 2
476 0569 2 IF NOT .status           ! Probably ss$_endoftape
477 0570 2 THEN
478 0571 2 BEGIN
479 0572 2 cur = .base + (.blks_to_write - .bc); ! Set cur to high block written before error
480 0573 2 base = .cur;           ! Say that base is the current
481 0574 2 ctx [ctx$l_cur_byte] = 0; ! Say that no bytes in last block
482 0575 2 exch$io_dos11_skip_record (.volb, -1); ! Backup one record
483 0576 2 RETURN .status;     ! Return the error status
484 0577 2 END;

```

```

485 0578 4
486 0579 4      ! Adjust our pointers
487 0580 4
488 0581 4      bp = .bp + 512;          ! Move to the next block
489 0582 4      bc = .bc - 1;
490 0583 4      IF .bc LEQ 0 THEN EXITLOOP;      ! Exit if all have been read
491 0584 4
492 0585 4      END;
493 0586 4      END;
494 0587 4
495 0588 4      ! If we have exceeded the previous high water mark, save the new mark
496 0589 4
497 0590 4      temp = (.base + (.blks_to_write-1));
498 0591 4      IF .temp GTRU .ctx [ctx$l_high_block_written]
499 0592 4      THEN
500 0593 4          ctx [ctx$l_high_block_written] = .temp;
501 0594 4
502 0595 4      ! Move the good data to the start of the buffer
503 0596 4
504 0597 4      CH$MOVE (.buf_len, .buf_start, .buf);
505 0598 4
506 0599 4      ! Change the base pointer to show what we just did, buf_high_block is still valid
507 0600 4
508 0601 4      base = .cur;
509 0602 4
510 0603 4      ! Change the high block pointer to show what we just did
511 0604 4
512 0605 4      high = MINU ((.high + .blks_to_write), .eof);
513 0606 4
514 0607 4      $trace_print_lit ('context at exit');
515 0608 4      $$show_context;
516 0609 4
517 0610 4      RETURN true;
518 0611 4
519 0612 4      END;

```

				OFFC 0000	.ENTRY	PDP BUFFER ADVANCE WRITE, Save R2,R3,R4,R5,-;	0407
						R6,R7,R8,R9,R10,R11	
		5E	04	C2 00002	SUBL2	#4, SP	
		58	04	AC D0 00005	MOVL	CTX, R8	0453
		5A	2C	A8 9E 00009	MOVAB	44(R8), R10	
		7E	01CA	8F 3C 0000D	MOVZWL	#458, -(SP)	0465
				58 DD 00012	PUSHL	R8	
	00000000G	00		02 FB 00014	CALLS	#2, PDP_CHECK_CTX	
		6A	1C	A8 D1 0001B	CMPL	28(R8), (R10)	0469
				07 1F 0001F	BLSSU	1\$	
	20	A8	30	A8 D1 00021	CMPL	48(R8), 32(R8)	
				13 1B 00026	BLEQU	2\$	
		7E	F2	8F 9A 00028 1\$:	MOVZBL	#242, -(SP)	
				01 DD 0002C	PUSHL	#1	
				8F DD 0002E	PUSHL	#EXCH\$ BADLOGIC	
	00000000G	00		03 FB 00034	CALLS	#3, LIB\$STOP	
	52	1C	A8	6A C3 0003B 2\$:	SUBL3	(R10), 28(R8), R2	0473

		56		52	DO	00040		MOVL	R2, BLKS_TO_WRITE		
		59	18	A8	DO	00043		MOVL	24(R8), R9	0478	
				13	12	00047		BNEQ	3\$		
		7E	C2	8F	9A	00049		MOVZBL	#194, -(SP)		
				01	DD	0004D		PUSHL	#1		
			00000000G	8F	DD	0004F		PUSHL	#EXCHS_BADLOGIC		
50	00000000G	00		03	FB	00055		CALLS	#3, LIB\$STOP		
58		52		09	78	0005C	3\$:	ASHL	#9, R2, R0	0479	
50	30	50		59	C1	00060		ADDL3	R9, R0, BUF_START		
50		A8		6A	C3	00064		SUBL3	(R10), 48(R8), R0	0480	
50		50		09	78	00069		ASHL	#9, R0, R0		
		50	0200	C0	49	9E	0006D	MOVAB	512(R0)[R9], BUF_END		
57		50		5B	C3	00073		SUBL3	BUF_START, BUF_END, BUF_LEN	0481	
	00010000	8F		57	D1	00077		CMPL	BUF_LEN, #65536	0482	
				13	1F	0007E		BLSSU	4\$		
		7E	AD	8F	9A	00080		MOVZBL	#173, -(SP)		
				01	DD	00084		PUSHL	#1		
			00000000G	8F	DD	00086		PUSHL	#EXCHS_BADLOGIC		
2C	00000000G	00		03	FB	0008C		CALLS	#3, LIB\$STOP		
	28	A8		02	E1	00093	4\$:	BBC	#2, 40(R8), 6\$	0486	
				24	A8	D5	00098	TSTL	36(R8)	0488	
				0F	13	0009B		BEQL	5\$		
				56	D6	0009D		INCL	BLKS_TO_WRITE	0492	
50		57		24	A8	C3	0009F	SUBL3	36(R8), -BUF_LEN, R0	0494	
00		6E		00	2C	000A4		MOVCS	#0, (SP), #0, R0, @36(R8)[BUF_START]		
				24	B8	4B	000A9				
13	28	A8		02	E1	000AC	5\$:	BBC	#2, 40(R8), 6\$	0499	
	FFFFFFF	8F		20	A8	D1	000B1	CMPL	32(R8), #-1	0501	
				09	12	000B9		BNEQ	6\$		
50		6A		56	C1	000BB		ADDL3	BLKS_TO_WRITE, (R10), R0	0506	
	20	A8		A0	9E	000BF		MOVAB	-1(R0), 32(R8)		
				56	D5	000C4	6\$:	TSTL	BLKS_TO_WRITE	0515	
				03	12	000C6		BNEQ	7\$		
				00	C5	31	000C8	BRW	17\$		
		0C		56	D1	000CB	7\$:	CMPL	BLKS_TO_WRITE, #12	0521	
				13	1B	000CE		BLEQU	8\$		
		7E	AE	8F	9A	000D0		MOVZBL	#174, -(SP)		
				01	DD	000D4		PUSHL	#1		
			00000000G	8F	DD	000D6		PUSHL	#EXCHS_BADLOGIC		
		00		03	FB	000DC		CALLS	#3, LIB\$STOP		
		52	14	A8	DO	000E3	8\$:	MOVL	20(R8), R2	0522	
		03	58	A2	91	000E7		CMPB	88(R2), #3		
				21	12	000EB		BNEQ	9\$		
			0240	8F	BB	000ED		PUSHR	#*M<R6,R9>	0528	
				6A	DD	000F1		PUSHL	(R10)	0527	
				52	DD	000F3		PUSHL	R2	0526	
		00000000G		04	FB	000F5		CALLS	#4, EXCHSIO_RT11_WRITE		
		54		50	DO	000FC		MOVL	R0, STATUS		
		63		54	E8	000FF		B_LBS	STATUS, 14\$		
				A8	DD	00102		PUSHL	16(R8)	0532	
		00000000G		01	FB	00105		CALLS	#1, EXCHSRT11_BAD_FILE		
				4B	11	0010C		BRB	12\$	0533	
				8F	3C	0010E	9\$:	MOVZWL	#512, BL	0543	
		6E	0200	56	DO	00113		MOVL	BLKS_TO_WRITE, BC	0544	
		53		59	DO	00116		MOVL	R9, BP	0545	
0E	28	A8		02	E1	00119	10\$:	BBC	#2, 40(R8), 11\$	0553	
		01		53	D1	0011E		CMPL	BC, #1	0555	

EXCHSPDP
V04-000

Small PDP-11 record structure routines
pdp_buffer_advance_write (ctx)

N 8
16-Sep-1984 01:11:46
14-Sep-1984 12:29:07

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

Page 14
(4)

			09	12	00121	BNEQ	11\$			
			24	A8	D5 00123	TSTL	36(R8)			0557
			04	13	00126	BEQL	11\$			
		6E	24	A8	D0 00128	MOVL	36(R8), BL			0559
			6E	DD	0012C	11\$: PUSHL	BL			0565
			24	BB	0012E	PUSHR	#*M<R2,R5>			0563
	00000000G	EF	03	FB	00130	CALLS	#3, EXCH\$IO_DOS11_WRITE			
		54	50	D0	00137	MOVL	R0, STATUS			
		20	54	E8	0013A	BLBS	STATUS, 13\$			0569
1C	50	56	53	C3	0013D	SUBL3	BC, BLKS_TO_WRITE, R0			0572
	AB	50	6A	C1	00141	ADDL3	(R10), R0, 28(R8)			
		6A	1C	A8	D0 00146	MOVL	28(R8), (R10)			0573
			24	A8	D4 0014A	CLRL	36(R8)			0574
		7E	01	CE	0014D	MNEGL	#1, -(SP)			0575
			52	DD	00150	PUSHL	R2			
	00000000G	EF	02	FB	00152	CALLS	#2, EXCH\$IO_DOS11_SKIP_RECORD			
		50	54	D0	00159	12\$: MOVL	STATUS, R0			0576
				04	0015C	RET				
		55	0200	C5	9E 0015D	13\$: MOVAB	512(R5), BP			0581
		B4		53	F5 00162	SOBGTR	BC, 10\$			0582
		6A		56	C1 00165	14\$: ADDL3	BLKS_TO_WRITE, (R10), R0			0590
50				50	D7 00169	DECL	TEMP			
		34	A8	50	D1 0016B	CMPL	TEMP, 52(R8)			0591
				04	1B 0016F	BLEQU	15\$			
		34	A8	50	D0 00171	MOVL	TEMP, 52(R8)			0593
				57	28 00175	15\$: MOV3	BUF_LEN, (BUF_START), (R9)			0597
69		6B	1C	A8	D0 00179	MOVL	28(R8), (R10)			0601
		6A	30	A8	C1 0017D	ADDL3	48(R8), BLKS_TO_WRITE, R0			0605
		56		50	D1 00182	CMPL	R0, 32(R8)			
		20	A8	04	1B 00186	BLEQU	16\$			
				A8	D0 00188	MOVL	32(R8), R0			
		30	50	D0	0018C	16\$: MOVL	R0, 48(R8)			
			50	D0	00190	17\$: MOVL	#1, R0			0610
			01	D0	00193	RET				0612

; Routine Size: 404 bytes, Routine Base: EXCHSPDP_CODE + 016F


```
.. 521 0613 1 GLOBAL ROUTINE pdp_buffer_check (ctx : $ref_bblock, out_filb : $ref_bblock) : jsb_r2r3 = %SBTTL 'pdp_
.. 522 0614 2 BEGIN
.. 523 0615 2 ++
.. 524 0616 2
.. 525 0617 2 FUNCTIONAL DESCRIPTION:
.. 526 0618 2
.. 527 0619 2 Handle the situation of buffer overflow by either writing some blocks or signalling EOF.
.. 528 0620 2
.. 529 0621 2 INPUTS:
.. 530 0622 2
.. 531 0623 2 ctx - Output file context block
.. 532 0624 2 out_filb - Output file block
.. 533 0625 2
.. 534 0626 2 IMPLICIT INPUTS:
.. 535 0627 2
.. 536 0628 2 none
.. 537 0629 2
.. 538 0630 2 OUTPUTS:
.. 539 0631 2
.. 540 0632 2 none
.. 541 0633 2
.. 542 0634 2 IMPLICIT OUTPUTS:
.. 543 0635 2
.. 544 0636 2 none
.. 545 0637 2
.. 546 0638 2 ROUTINE VALUE:
.. 547 0639 2
.. 548 0640 2 true if success, false if any error
.. 549 0641 2
.. 550 0642 2 SIDE EFFECTS:
.. 551 0643 2
.. 552 0644 2 error conditions will be signaled
.. 553 0645 2 --
.. 554 0646 2
.. 555 0647 2 $dbgtrc_prefix ('pdp_buffer_check> ');
.. 556 0648 2
.. 557 0649 2 REGISTER
.. 558 0650 2 tmp
.. 559 0651 2 ;
.. 560 0652 2
.. 561 0653 2 $debug_print_lit ('entry');
.. 562 0654 2
.. 563 0655 2 ! If the EOF block is in the buffer
.. 564 0656 2
.. 565 0657 2 IF .ctx [ctx$l_buf_high_block] GEQU .ctx [ctx$l_eof_block]
.. 566 0658 2 THEN
.. 567 0659 2
.. 568 0660 2 ! Don't have any more room at the inn
.. 569 0661 2
.. 570 0662 2 $exch_signal_return (exch$_rtouteof, 2, .out_filb [filb$l_result_name_len], out_filb [filb$_result_name
.. 571 0663 2
.. 572 0664 2 ! Otherwise, write some data and recursively retry the put
.. 573 0665 2
.. 574 0666 2 ELSE
.. 575 0667 2 BEGIN
.. 576 0668 2 IF NOT (tmp = pdp_buffer_advance_write (.ctx))
.. 577 0669 2 THEN
```

```

: 578      0670 3      RETURN .tmp;
: 579      0671 3      RETURN exch$pdp_put ();
: 580      0672 2      END;
: 581      0673 2
: 582      0674 1 END;

```

! And then try it again

```

                                .EXTRN EXCH$_RTOUTEOF
                                PDP_BUFFER_CHECK::
20  A2      30  A2  D1 0000  Cmpl      48(CTX), 32(CTX)      : 0657
                                Blssu      1$
                                52 00000000G 8F D0 00007  Movl      #EXCH$_RTOUTEOF, TEMP      : 0662
                                5A  A3  9F 0000E  Pushab   90(OUT_FILB)
                                3A  A3  DD 00011  Pushl    58(OUT_FILB)
                                02  D7 00014  Pushl    #2
                                52  DD 00016  Pushl    TEMP
00000000G 00  04  FB 00018  Calls    #4, LIB$SIGNAL
                                50  52  D0 0001F  Movl     TEMP, R0      : 0667
                                05 00022  Rsb
                                52  DD 00023  1$:  Pushl    CTX      : 0668
                                FE42 CF  01  FB 00025  Calls    #1, PDP_BUFFER_ADVANCE_WRITE
                                05  50  E9 0002A  Blbc     TMP, 2$
                                0000V CF  00  FB 0002D  Calls    #0, EXCH$PDP_PUT      : 0671
                                05 00032  2$:  Rsb      : 0674

```

: Routine Size: 51 bytes, Routine Base: EXCH\$PDP_CODE + 0303

```
584 0675 1 GLOBAL ROUTINE pdp_buffer_update (ctx : $ref_bblock, next_buf) : jsb_r2r3 = %SBTTL 'pdp_buffer_update'
585 0676 2 BEGIN
586 0677 2 ++
587 0678 2
588 0679 2 FUNCTIONAL DESCRIPTION:
589 0680 2
590 0681 2 Update the current byte information in the context
591 0682 2
592 0683 2 INPUTS:
593 0684 2
594 0685 2 ctx - Output file context block
595 0686 2 next_buf - New current record pointer
596 0687 2
597 0688 2 IMPLICIT INPUTS:
598 0689 2
599 0690 2 none
600 0691 2
601 0692 2 OUTPUTS:
602 0693 2
603 0694 2 none
604 0695 2
605 0696 2 IMPLICIT OUTPUTS:
606 0697 2
607 0698 2 none
608 0699 2
609 0700 2 ROUTINE VALUE:
610 0701 2
611 0702 2 true if success, false if any error
612 0703 2
613 0704 2 SIDE EFFECTS:
614 0705 2
615 0706 2 error conditions will be signaled
616 0707 2 --
617 0708 2
618 0709 2 $dbgtrc_prefix ('pdp_buffer_update> ');
619 0710 2
620 0711 2 REGISTER
621 0712 2 five12,
622 0713 2 tmp
623 0714 2 ;
624 0715 2
625 0716 2 $debug_print_lit ('entry');
626 0717 2
627 0718 2 ! Update the next record position
628 0719 2
629 0720 2 five12 = 512;
630 0721 2 $logic_check (2, (.ctx [ctx$a_buffer] NEQ 0), 201);
631 0722 2 tmp = .next_buf - .ctx [ctx$a_buffer]; ! Save the updated position for the next put
632 0723 2 ctx [ctx$l_cur_byte] = .tmp MOD .five12;
633 0724 2 ctx [ctx$l_cur_block] = (.tmp / .five12) + .ctx [ctx$l_buf_base_block];
634 0725 2
635 0726 2 RETURN true;
636 0727 2
637 0728 1 END;
```

			53	DD	00000	PDP_BUFFER_UPDATE::			
						PUSHL	R3	: 0675	
		53	0200	8F	3C	00002	MOVZWL	#512, FIVE12 : 0720	
			18	A2	D5	00007	TSTL	24(CTX) : 0721	
				13	12	0000A	BNEQ	1\$	
		7E	C9	8F	9A	0000C	MOVZBL	#201, -(SP)	
				01	DD	00010	PUSHL	#1	
				8F	DD	00012	PUSHL	#EXCH\$ BADLOGIC	
	00000000G	00		03	FB	00018	CALLS	#3, LIB\$STOP	
		6E	18	A2	C3	0001F	1\$:	SUBL3	24(CTX), NEXT_BUF, TMP : 0722
7E	50			01	7A	00024	EMUL	#1, TMP, #0, =(SP) : 0723	
51	00			53	7B	00029	EDIV	FIVE12, (SP)+, R1, R1	
		8E		51	D0	0002E	MOVL	R1, 36(CTX)	
		24		53	C6	00032	DIVL2	FIVE12, R0 : 0724	
		50		9E	00035	MOVAB	@44(CTX)[R0], 28(CTX)		
		1C		01	D0	0003B	MOVL	#1, R0 : 0726	
		50		04	C0	0003E	ADDL2	#4, SP : 0728	
		5E		05	00041	RSB		:	

; Routine Size: 66 bytes, Routine Base: EXCH\$PDP_CODE + 0336

```
639 0729 1 GLOBAL ROUTINE pdp_check_ctx (ctx : $ref_bblock, code) : NOVALUE = %SBTTL 'pdp_check_ctx'
640 0730 2 BEGIN
641 0731 2 +-
642 0732 2
643 0733 2 FUNCTIONAL DESCRIPTION:
644 0734 2
645 0735 2     Check for a valid context block
646 0736 2
647 0737 2 INPUTS:
648 0738 2
649 0739 2     ctx      - Output file context block
650 0740 2     code     - Error code to use if the check fails
651 0741 2
652 0742 2 IMPLICIT INPUTS:
653 0743 2
654 0744 2     none
655 0745 2
656 0746 2 OUTPUTS:
657 0747 2
658 0748 2     none
659 0749 2
660 0750 2 IMPLICIT OUTPUTS:
661 0751 2
662 0752 2     none
663 0753 2
664 0754 2 ROUTINE VALUE:
665 0755 2
666 0756 2     none
667 0757 2
668 0758 2 SIDE EFFECTS:
669 0759 2
670 0760 2     error conditions will be signaled
671 0761 2 --
672 0762 2
673 0763 2 $dbgtrc_prefix ('pdp_check_ctx> ');
674 0764 2
675 0765 2 LOCAL
676 0766 2     size,
677 0767 2     type
678 0768 2     ;
679 0769 2
680 0770 2 BIND
681 0771 2     filb = ctx [ctx$a_assoc_filb]      : $ref_bblock,
682 0772 2     volb = ctx [ctx$a_assoc_volb]     : $ref_bblock
683 0773 2     ;
684 0774 2
685 0775 2 $debug_print_lit ('entry');
686 0776 2
687 0777 2 ! The context block must exist
688 0778 2
689 0779 2 IF .ctx EQL 0
690 0780 2 THEN
691 0781 2     $exch_signal_stop (exch$_blockcheck0, 1, .code);
692 0782 2
693 0783 2 ! Now look for either an RT11CTX block or a DOS11CTX block
694 0784 2
695 0785 2 IF .ctx [ctx$b_type] EQL exchblk$_rt11ctx
```

```

696 0786 2 THEN
697 0787 BEGIN
698 0788 IF .ctx [ctx$w_size] NEQ exchblk$s_rt11ctx
699 0789 THEN
700 0790 BEGIN
701 0791 size = exchblk$s_rt11ctx;
702 0792 type = exchblk$k_rt11ctx;
703 0793 $exch_signal_stop (exch$_blockcheck, 6, .code, .ctx, .ctx [ctx$w_size], .size, .ctx [ctx$b_type], .t
704 0794 END;
705 0795 END
706 0796 ELSE IF .ctx [ctx$b_type] EQL exchblk$k_dos11ctx
707 0797 THEN
708 0798 BEGIN
709 0799 IF .ctx [ctx$w_size] NEQ exchblk$s_dos11ctx
710 0800 THEN
711 0801 BEGIN
712 0802 size = exchblk$s_dos11ctx;
713 0803 type = exchblk$k_dos11ctx;
714 0804 $exch_signal_stop (exch$_blockcheck, 6, .code, .ctx, .ctx [ctx$w_size], .size, .ctx [ctx$b_type], .t
715 0805 END;
716 0806 END
717 0807 ELSE
718 0808 BEGIN
719 0809 size = exchblk$s_rt11ctx;
720 0810 type = exchblk$k_rt11ctx;
721 0811 $exch_signal_stop (exch$_blockcheck, 6, .code, .ctx, .ctx [ctx$w_size], .size, .ctx [ctx$b_type], .type)
722 0812 END;
723 0813
724 0814 IF .filb EQL 0
725 0815 THEN
726 0816 $exch_signal_stop (exch$_blockcheck0, 1, (10000+.code));
727 0817
728 0818 IF .filb [filb$w_size] NEQ exchblk$s_filb
729 0819 OR
730 0820 .filb [filb$b_type] NEQ exchblk$k_filb
731 0821 THEN
732 0822 $exch_signal_stop (exch$_blockcheck, 6, (10000+.code), .filb,
P 0823 .filb [filb$w_size], exchblk$s_filb,
734 0824 .filb [filb$b_type], exchblk$k_filb);
735 0825
736 0826 IF .volb EQL 0
737 0827 THEN
738 0828 $exch_signal_stop (exch$_blockcheck0, 1, (20000+.code));
739 0829
740 0830 IF .volb [volb$w_size] NEQ exchblk$s_volb
741 0831 OR
742 0832 .volb [volb$b_type] NEQ exchblk$k_volb
743 0833 THEN
P 0834 $exch_signal_stop (exch$_blockcheck, 6, (20000+.code), .volb,
P 0835 .volb [volb$w_size], exchblk$s_volb,
746 0836 .volb [volb$b_type], exchblk$k_volb);
747 0837
748 0838 1 END;

```

.EXTRN EXCH\$_BLOCKCHECK0

				.EXTRN	LIB\$STOP, EXCH\$_BLOCKCHECK	
			001C 00000	.ENTRY	PDP CHECK_CTX, Save R2,R3,R4	: 0729
	54	00000000G	00 9E 00002	MOVAB	LIB\$STOP, R4	: 0771
	51	04	AC D0 00009	MOVL	CTX, R1	: 0779
			05 12 0000D	BNEQ	1\$: 0781
		08	AC DD 0000F	PUSHL	CODE	: 0785
			52 11 00012	BRB	6\$: 0788
	53	0A	A1 9A 00014 1\$:	MOVZBL	10(R1), R3	: 0791
F4	8F		53 91 00018	CMPB	R3, #244	: 0796
			0A 12 0001C	BNEQ	2\$: 0799
0082	8F	08	A1 B1 0001E	CMPW	8(R1), #130	: 0802
			31 13 00024	BEQL	5\$: 0803
			18 11 00026	BRB	3\$: 0804
	FC	8F	53 91 00028 2\$:	CMPB	R3, #252	: 0809
			12 12 0002C	BNEQ	3\$: 0810
008A	8F	08	A1 B1 0002E	CMPW	8(R1), #138	: 0811
			21 13 00034	BEQL	5\$: 0814
	52	8A	8F 9A 00036	MOVZBL	#138, SIZE	: 0816
	50	FC	8F 9A 0003A	MOVZBL	#252, TYPE	: 0818
			08 11 0003E	BRB	4\$: 0820
	52	82	8F 9A 00040 3\$:	MOVZBL	#130, SIZE	: 0824
	50	F4	8F 9A 00044	MOVZBL	#244, TYPE	: 0826
			50 DD 00048 4\$:	PUSHL	TYPE	: 0828
			0C BB 0004A	PUSHR	#*M<R2,R3>	: 0830
	7E	08	A1 3C 0004C	MOVZWL	8(R1), -(SP)	: 0832
			51 DD 00050	PUSHL	R1	: 0836
		08	AC DD 00052	PUSHL	CODE	: 0838
			3C 11 00055	BRB	9\$: 0840
	50	10	A1 D0 00057 5\$:	MOVL	16(R1), R0	: 0842
			0B 12 0005B	BNEQ	7\$: 0844
7E	08	AC 00002710	8F C1 0005D	ADDL3	#10000, CODE, -(SP)	: 0846
			3C 11 00066 6\$:	BRB	11\$: 0848
035B	8F	08	A0 B1 00068 7\$:	CMPW	8(R0), #859	: 0850
			07 12 0006E	BNEQ	8\$: 0852
	FA	8F	0A A0 91 00070	CMPB	10(R0), #250	: 0854
			1E 13 00075	BEQL	10\$: 0856
	7E	FA	8F 9A 00077 8\$:	MOVZBL	#250, -(SP)	: 0858
	7E	0A	A0 9A 0007B	MOVZBL	10(R0), -(SP)	: 0860
	7E	035B	8F 3C 0007F	MOVZWL	#859, -(SP)	: 0862
	7E	08	A0 3C 00084	MOVZWL	8(R0), -(SP)	: 0864
			50 DD 00088	PUSHL	R0	: 0866
7E	08	AC 00002710	8F C1 0008A	ADDL3	#10000, CODE, -(SP)	: 0868
			46 11 00093 9\$:	BRB	14\$: 0870
	50	14	A1 D0 00095 10\$:	MOVL	20(R1), R0	: 0872
			15 12 00099	BNEQ	12\$: 0874
7E	08	AC 00004E20	8F C1 0009B	ADDL3	#20000, CODE, -(SP)	: 0876
			01 DD 000A4 11\$:	PUSHL	#1	: 0878
		00000000G	8F DD 000A6	PUSHL	#EXCH\$_BLOCKCHECK0	: 0880
	64		03 FB 000AC	CALLS	#3, LIB\$STOP	: 0882
			04 000AF	RET		: 0884
041B	8F	08	A0 B1 000B0 12\$:	CMPW	8(R0), #1051	: 0886
			07 12 000B6	BNEQ	13\$: 0888
	F3	8F	0A A0 91 000B8	CMPB	10(R0), #243	: 0890
			27 13 000BD	BEQL	15\$: 0892
	7E	F3	8F 9A 000BF 13\$:	MOVZBL	#243, -(SP)	: 0894
	7E	0A	A0 9A 000C3	MOVZBL	10(R0), -(SP)	: 0896

EXCH\$PDP
V04-000

Small PDP-11 record structure routines
pdp_check_ctx

I 9
16-Sep-1984 01:11:46
14-Sep-1984 12:29:07

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

Page 22
(7)

		7E	041B	8F	3C	000C7		MOVZWL	#1051, -(SP)	
		7E	08	A0	3C	000CC		MOVZWL	8(R0), -(SP)	
				50	DD	000D0		PUSHL	R0	:
	7E	08	AC 00004E20	8F	C1	000D2		ADDL3	#20000, CODE, -(SP)	:
				06	DD	000DB	14\$:	PUSHL	#6	:
			00000000G	8F	DD	000DD		PUSHL	#EXCH\$ BLOCKCHECK	:
		64		08	FB	000E3		CALLS	#8, LIB\$STOP	:
				04	000E6	15\$:		RET		: 0838

; Routine Size: 231 bytes, Routine Base: EXCH\$PDP_CODE + 0378

EXC
V04


```

: 750 0839 1 GLOBAL ROUTINE pdp_copy_binary_record (in_len, in_buf : $ref_bvector, %SBTTL 'pdp_copy_binary_record'
: 751 0840 1 out_buf : $ref_bvector) : NOVALUE =
: 752 0841 1 BEGIN
: 753 0842 1 ++
: 754 0843 1
: 755 0844 1 FUNCTIONAL DESCRIPTION:
: 756 0845 1
: 757 0846 1 Copy the input record to a buffer, reformatting it as a valid formatted-binary record.
: 758 0847 1
: 759 0848 1 INPUTS:
: 760 0849 1
: 761 0850 1 in_len - length of the input record
: 762 0851 1 in_buf - address of the input record
: 763 0852 1
: 764 0853 1 IMPLICIT INPUTS:
: 765 0854 1
: 766 0855 1 none
: 767 0856 1
: 768 0857 1 OUTPUTS:
: 769 0858 1
: 770 0859 1 out_buf - address of the output buffer which receives the formatted-binary copy of the input
: 771 0860 1
: 772 0861 1 IMPLICIT OUTPUTS:
: 773 0862 1
: 774 0863 1 none
: 775 0864 1
: 776 0865 1 ROUTINE VALUE:
: 777 0866 1
: 778 0867 1 none
: 779 0868 1
: 780 0869 1 SIDE EFFECTS:
: 781 0870 1
: 782 0871 1 none
: 783 0872 1 --
: 784 0873 1
: 785 0874 1 $dbgtrc_prefix ('pdp_copy_binary_record> ');
: 786 0875 1
: 787 0876 1 REGISTER
: 788 0877 1 ip, ! Input pointer
: 789 0878 1 op, ! Output pointer
: 790 0879 1 chksum : BYTE,
: 791 0880 1 neg_chksum : BYTE,
: 792 0881 1 char : BYTE ! Current character
: 793 0882 1
: 794 0883 1
: 795 0884 1 BIND
: 796 0885 1 sentinel = out_buf [0] : WORD, ! Sentinel word, first two bytes of the output
: 797 0886 1 length = out_buf [2] : WORD ! Length word, next two bytes
: 798 0887 1
: 799 0888 1
: 800 0889 1 $debug_print_fao ('entry, len=!UL, buf[0:19]='!AF'', .in_len, 20, .in_buf);
: 801 0890 1
: 802 0891 1 ! Initialize our local data segments
: 803 0892 1
: 804 0893 1 op = .out_buf; ! Output buffer pointer
: 805 0894 1 ip = .in_buf; ! Input pointer at the start of the record
: 806 0895 1 chksum = 0;

```

```

0896 2
0897 2 ! Put the sentinel and length words in the buffer
0898 2
0899 2 sentinel = 1;
0900 2 length = .in_len + 4;
0901 2
0902 2 ! Prepare the checksum from the first four bytes
0903 2
0904 2 DECR c FROM 3 TO 0
0905 2 DO
0906 2     chksum = .chksum + CH$RCHAR_A (op);
0907 2
0908 2 ! Start grabbing bytes
0909 2
0910 2 IF .in_len GTRU 0
0911 2 THEN
0912 2     DECR c FROM .in_len-1 TO 0
0913 2     DO
0914 2         BEGIN
0915 2
0916 2         char = CH$RCHAR_A (ip);           ! Read the new character and advance the input pointer
0917 2
0918 2         chksum = .chksum + .char;         ! Add this byte to the checksum
0919 2
0920 2         CH$WCHAR_A (.char, op);          ! Move it to the output and advance the output pointer
0921 2
0922 2         END;
0923 2
0924 2 ! Store the negated checksum
0925 2
0926 2 neg_chksum = -.chksum;
0927 2 CH$WCH, ? (.neg_chksum, .op);           ! Move it to the output
0928 2
0929 2 RETURN;
0930 1 END;

```

53	0C	AC	08	02	C1	00002	.ENTRY	PDP_COPY_BINARY_RECORD, Save R2,R3,R4	0839
		50		AC	7D	00007	ADDL3	#2, -OUT_BUF, R3	0886
				52	94	0000B	MOVQ	IN_BUF, IP	0894
				01	B0	0000D	CLRB	CH\$SUM	0895
63	0C	BC		04	A1	00011	MOVW	#1, @OUT_BUF	0899
	04	AC		03	D0	00016	ADDW3	#4, IN_LEN, (R3)	0900
		53		81	9A	00019	MOVL	#3, C	0904
		54		54	80	0001C	MOVZBL	(OP)+, R4	0906
		F7		53	F4	0001F	ADDB2	R4, CH\$SUM	
			04	12	13	00025	SOBGEQ	C, 1\$	
				09	11	0002B	TSTL	IN_LEN	0910
		54	04	09	D0	00027	BEQL	4\$	
				09	11	0002B	MOVL	IN_LEN, C	0912
		53		80	90	0002D	BRB	3\$	
		52		53	80	00030	MOVB	(IP)+, CHAR	0916
		81		53	90	00033	ADDB2	CHAR, CH\$SUM	0918
							MOVB	CHAR, (OP)+	0920

EXCHSPDP
V04-000

Small PDP-11 record structure routines
pdp_copy_binary_record

L⁹
16-Sep-1984 01:11:46
14-Sep-1984 12:29:07

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

Page 25
(8)

F4	54	F4	00036	3\$:	SOBGEQ	C, 2\$:	0912
50	52	8E	00039	4\$:	MNEGB	CHKSUM, NEG_CHKSUM	:	0926
61	50	90	0003C		MOVB	NEG_CHKSUM, (OP)	:	0927
	04	0003F			RET		:	0930

; Routine Size: 64 bytes, Routine Base: EXCHSPDP_CODE + 045F

```

843 0931 1 GLOBAL ROUTINE pdp_copy_stream_record (in_len, in_buf : $ref_bvector, %SBTTL 'pdp_copy_stream_record'
844 0932 1                                     out_buf : $ref_bvector) =
845 0933 2 BEGIN
846 0934 2 ++
847 0935 2
848 0936 2 FUNCTIONAL DESCRIPTION:
849 0937 2
850 0938 2     Copy the input record to a buffer, reformatting it as a valid stream format record. The length of t
851 0939 2     output record is returned.
852 0940 2
853 0941 2 INPUTS:
854 0942 2
855 0943 2     in_len - length of the input record
856 0944 2     in_buf - address of the input record
857 0945 2
858 0946 2 IMPLICIT INPUTS:
859 0947 2
860 0948 2     none
861 0949 2
862 0950 2 OUTPUTS:
863 0951 2
864 0952 2     out_buf - address of the output buffer which receives the stream format copy of the input, including
865 0953 2     record terminator(s)
866 0954 2
867 0955 2 IMPLICIT OUTPUTS:
868 0956 2
869 0957 2     none
870 0958 2
871 0959 2 ROUTINE VALUE:
872 0960 2
873 0961 2     The length of the output record, including terminator
874 0962 2
875 0963 2 SIDE EFFECTS:
876 0964 2
877 0965 2     none
878 0966 2 --
879 0967 2
880 0968 2 $dbgtrc_prefix ('pdp_copy_stream_record> ');
881 0969 2
882 0970 2 REGISTER
883 0971 2     ip,           ! Input pointer
884 0972 2     op,           ! Output pointer
885 0973 2     ol,           ! Output length
886 0974 2     char          : BYTE ! Current character
887 0975 2     ;
888 0976 2
889 0977 2 $debug_print_fao ('entry, len=!UL, buf[0:19]='!AF'', .in_len, 20, .in_buf);
890 0978 2
891 0979 2 ! Initialize our local data segments
892 0980 2
893 0981 2 op = .out_buf;      ! Output buffer pointer
894 0982 2 ip = .in_buf;      ! Input pointer at the start of the record
895 0983 2 char = 0;          ! Preset for the later test, in case 0 length input
896 0984 2
897 0985 2 ! Start grabbing bytes
898 0986 2
899 0987 2 IF .in_len GTRU 0

```

```

900 0988 2 THEN
901 0989     DECR c FROM .in_len-1 TO 0
902 0990     DO
903 0991         BEGIN
904 0992             ! Read the character and clear the high bit
905 0993             !
906 0994             !
907 0995             char = CH$RCHAR_A (ip);           ! Read the new character and advance the input pointer
908 0996             char <7,1,0> = 0;             ! Clear the high bit
909 0997             !
910 0998             ! Now look at the character and do something with it
911 0999             !
912 1000             SELECTONEU .char OF
913 1001             SET
914 1002                 [NUL, DEL, VT] :
915 1003                     ;
916 1004                 [OTHERWISE] :
917 1005                     CH$WCHAR_A (.char, op);
918 1006             ;
919 1007             ;
920 1008             ;
921 1009             TES;
922 1010             ;
923 1011             END;
924 1012             ;
925 1013             ! If the final char was either a form feed or a line feed, we are done. Otherwise add the <CR><LF> pair
926 1014             !
927 1015             IF ((.char NEQ LF)           ! line feed
928 1016             AND
929 1017             (.char NEQ FF))           ! form feed
930 1018             THEN
931 1019                 BEGIN
932 1020                 CH$WCHAR_A (CR, op);
933 1021                 CH$WCHAR_A (LF, op);
934 1022                 END;
935 1023             ;
936 1024             ! Calculate the final length
937 1025             !
938 1026             ol = .op - .out_buf;
939 1027             !
940 1028             $debug_print_fao ('output len !UL, record[0:19] '!AF'', .ol, 20, .out_buf);
941 1029             !
942 1030             RETURN .ol;
943 1031             END;

```

			000C 0000	.ENTRY	PDP_COPY_STREAM_RECORD, Save R2,R3	: 0931
50	08	AC	7D 00002	MOVQ	IN_BUF, IP	: 0982
		52	94 00006	CLRB	CHAR	: 0983
	04	AC	D5 00008	TSTL	IN_LEN	: 0987
		20	13 0000B	BEQL	3\$-	: 0989
53	04	AC	D0 0000D	MOVL	IN_LEN, C	: 0989
		17	11 00011	BRB	2\$-	: 0995
52		80	90 00013 1\$:	MOVB	(IP)+, CHAR	: 0995

	52	80	8F	8A	00016		BICB2	#128, CHAR	:	0996
			0E	13	0001A		BEQL	2\$:	1003
	0B		52	91	0001C		CMPB	CHAR, #11	:	
			09	13	0001F		BEQL	2\$:	
	7F	8F	52	91	00021		CMPB	CHAR, #127	:	
			03	13	00025		BEQL	2\$:	
	81		52	90	00027		MOVB	CHAR, (OP)+	:	1007
	E6		53	F4	0002A	2\$:	SOBGEQ	C, 1\$:	0989
	0A		52	91	0002D	3\$:	CMPB	CHAR, #10	:	1015
			0A	13	00030		BEQL	4\$:	
	0C		52	91	00032		CMPB	CHAR, #12	:	1017
			05	13	00035		BEQL	4\$:	
	81	0A0D	8F	B0	00037		MOVW	#2573, (OP)+	:	1020
50	51	0C	AC	C3	0003C	4\$:	SUBL3	OUT_BUF, OP, OL	:	1026
			04	00041			RET		:	1031

: Routine Size: 66 bytes, Routine Base: EXCH\$PDP_CODE + 049F

```

: 945 1032 1 GLOBAL ROUTINE exch$dpd_filter_filename (nam_len, nam_start) = %SBTTL 'exch$dpd_filter_filename'
: 946 1033 BEGIN
: 947 1034 ++
: 948 1035
: 949 1036 FUNCTIONAL DESCRIPTION:
: 950 1037
: 951 1038 Scan filename, removing characters which are invalid. The string will be modified in place.
: 952 1039
: 953 1040 INPUTS:
: 954 1041
: 955 1042 nam_len - length of the name
: 956 1043 nam_start - starting address of the filename
: 957 1044
: 958 1045 IMPLICIT INPUTS:
: 959 1046
: 960 1047 none
: 961 1048
: 962 1049 OUTPUTS:
: 963 1050
: 964 1051 the name string is modified in place
: 965 1052
: 966 1053 IMPLICIT OUTPUTS:
: 967 1054
: 968 1055 none
: 969 1056
: 970 1057 ROUTINE VALUE:
: 971 1058
: 972 1059 none
: 973 1060
: 974 1061 SIDE EFFECTS:
: 975 1062
: 976 1063 none
: 977 1064 --
: 978 1065
: 979 1066 $dbgtrc_prefix ('exch$dpd_filter_filename> ');
: 980 1067
: 981 1068
: 982 1069 REGISTER
: 983 1070 ip, ! Input pointer
: 984 1071 op, ! Output pointer
: 985 1072 char : BYTE ! Current character
: 986 1073 ;
: 987 1074
: 988 1075 $debug_print_lit ('entry');
: 989 1076
: 990 1077 IF (.nam_len EQL 0) ! Nothing to do in this case
: 991 1078 THEN
: 992 1079 RETURN .nam_len;
: 993 1080
: 994 1081 ! Initialize our local data segments
: 995 1082
: 996 1083 ip = .nam_start; ! Input pointer at the start of the buffer
: 997 1084 op = .ip; ! Output pointer starts at the beginning
: 998 1085
: 999 1086 DECR len FROM .nam_len - 1 TO 0
: 1000 1087 DO
: 1001 1088 BEGIN

```

```

: 1002      1089      | char = CH$RCHAR_A (ip);
: 1003      1090      | SELECTONE .char-OF
: 1004      1091      | SET
: 1005      1092      |     ['A' TO 'Z', '0' TO '9'] :
: 1006      1093      |
: 1007      1094      |     CH$WCHAR_A (.char, op);
: 1008      1095      |     [OTHERWISE] :
: 1009      1096      |     TES;
: 1010      1097      |     END;
: 1011      1098      | ! Return the length
: 1012      1099      | !
: 1013      1100      | RETURN .op - .nam_start;
: 1014      1101      |
: 1015      1102      | 1 END;

```

			000C 00000	.ENTRY	EXCH\$PDP_FILTER_FILENAME, Save R2,R3	: 1032
53	04	AC	D0 00002	MOVL	NAM_LEN, R3	: 1077
		04	12 00006	BNEQ	1\$	
50		53	D0 00008	MOVL	R3, R0	: 1079
			04 0000B	RET		
50	08	AC	D0 0000C 1\$:	MOVL	NAM_START, IP	: 1083
51		50	D0 00010	MOVL	IP, OP	: 1084
		1C	11 00013	BRB	5\$: 1086
52		80	90 00015 2\$:	MOVB	(IP)+, CHAR	: 1089
30		52	91 00018	CMPB	CHAR, #48	: 1092
		05	1F 0001B	BLSSU	3\$	
39		52	91 0001D	CMPB	CHAR, #57	
		0C	1B 00020	BLEQU	4\$	
41	8F	52	91 00022 3\$:	CMPB	CHAR, #65	
		09	1F 00026	BLSSU	5\$	
5A	8F	52	91 00028	CMPB	CHAR, #90	
		03	1A 0002C	BGTRU	5\$	
81		52	90 0002E 4\$:	MOVB	CHAR, (OP)+	: 1093
E1		53	F4 00031 5\$:	SOBGEQ	LEN, 2\$: 1086
51	08	AC	C2 00034	SUBL2	NAM_START, R1	: 1100
50		51	D0 00038	MOVL	R1, R0	: 1102
			04 0003B	RET		

: Routine Size: 60 bytes, Routine Base: EXCH\$PDP_CODE + 04E1


```

1017 1103 1 GLOBAL ROUTINE pdp_find_binary_record (filb : $ref_bblock, buf_start, %SBTTL 'pdp_find_binary_record'
1018 1104 1          buf_end : $ref_bvector, new_start) =
1019 1105 1 BEGIN
1020 1106 1 ++
1021 1107 1
1022 1108 1 FUNCTIONAL DESCRIPTION:
1023 1109 1
1024 1110 1     Scan buffer from start to end (if necessary) looking for a single formatted binary record. The addr
1025 1111 1     length of the record are copied to the record buffer pointers in the filb. The address of the next
1026 1112 1     unscanned byte is returned.
1027 1113 1
1028 1114 1 INPUTS:
1029 1115 1
1030 1116 1     filb      - pointer to the filb which contains the active record stream
1031 1117 1     buf_start - starting address in buffer to scan
1032 1118 1     buf_end   - one past the highest valid buffer address
1033 1119 1
1034 1120 1 IMPLICIT INPUTS:
1035 1121 1
1036 1122 1     none
1037 1123 1
1038 1124 1 OUTPUTS:
1039 1125 1
1040 1126 1     new_start - receives address of first unscanned byte
1041 1127 1
1042 1128 1 IMPLICIT OUTPUTS:
1043 1129 1
1044 1130 1     none
1045 1131 1
1046 1132 1 ROUTINE VALUE:
1047 1133 1
1048 1134 1     findbin$k_success - record 'placed' in filb, all is well
1049 1135 1     k_eob              - at end of buffer without finding complete record
1050 1136 1     k_bad_fmt         - problem with record format
1051 1137 1     k_too_big         - record exceeds length of output buffer
1052 1138 1     k_chksum          - computed checksum differs from stored checksum
1053 1139 1
1054 1140 1 SIDE EFFECTS:
1055 1141 1
1056 1142 1     none
1057 1143 1 --
1058 1144 1
1059 1145 1 $dbgtrc_prefix ('pdp_find_binary_record> ');
1060 1146 1
1061 1147 1
1062 1148 1 REGISTER
1063 1149 1     ip,          ! Input pointer
1064 1150 1     ol,          ! Output length
1065 1151 1     eob,         ! End of buffer
1066 1152 1     chksum       ! Check sum accumulator
1067 1153 1     neg_chksum   ! Negative of checksum for compares
1068 1154 1     char         ! Current character
1069 1155 1     ;
1070 1156 1
1071 1157 1 $debug_print_lit ('entry');
1072 1158 1 $block_check(2, .filb, filb, 495);
1073 1159 1

```

```
1074 1160 2 ! Initialize our local data segments
1075 1161 ~~~~~
1076 1162 ip = .buf_start; ! Input pointer at the start of the buifer
1077 1163 eob = .buf_end; ! End of buffer pointer one past the end of the buffer
1078 1164 ~~~~~
1079 1165 ! Skip any null bytes at the start of the record
1080 1166 ~~~~~
1081 1167 DO
1082 1168 BEGIN
1083 1169 ~~~~~
1084 1170 ! Check for the end of the input buffer. We make sure that the entire header is in the buffer
1085 1171 ~~~~~
1086 1172 IF .ip+4 GEQU .eob
1087 1173 THEN
1088 1174 RETURN findbin$k_eob;
1089 1175 ~~~~~
1090 1176 ! Read the character and advance the pointer
1091 1177 ~~~~~
1092 1178 char = CH$RCHAR_A (ip);
1093 1179 ~~~~~
1094 1180 END
1095 1181 ~~~~~
1096 1182 UNTIL .char NEQ 0;
1097 1183 ~~~~~
1098 1184 ! A formatted binary record has a word containing 1 followed by a word containing the length of the data + h
1099 1185 ~~~~~
1100 1186 IF (.char NEQ 1) OR (CH$RCHAR_A (ip) NEQ 0)
1101 1187 THEN
1102 1188 RETURN findbin$k_bad_fmt;
1103 1189 ~~~~~
1104 1190 ! Get the length, and initialize the checksum
1105 1191 ~~~~~
1106 1192 ol = (BIND len = .ip : WORD; .len) - 4; ! Interpret datum at input pointer as a word
1107 1193 chksum = 1 + CH$RCHAR_A (ip) + CH$RCHAR_A (ip); ! Checksum is 1 plus the two bytes of the length word
1108 1194 ~~~~~
1109 1195 ! Although we use locate mode, lets do a sanity check and refuse oversize records
1110 1196 ~~~~~
1111 1197 IF .ol GTRU filb$s_record_buffer
1112 1198 THEN
1113 1199 RETURN findbin$k_too_big;
1114 1200 ~~~~~
1115 1201 ! Make sure that the entire record plus the checksum byte are present in the buffer
1116 1202 ~~~~~
1117 1203 IF (.ip + .ol + 1) GEQU .eob
1118 1204 THEN
1119 1205 RETURN findbin$k_eob;
1120 1206 ~~~~~
1121 1207 ! Point the filb record information at the record we have found
1122 1208 ~~~~~
1123 1209 filb [filb$a_record] = ip;
1124 1210 filb [filb$l_record_len] = .ol;
1125 1211 ~~~~~
1126 1212 ! Calculate the checksum, then negate it
1127 1213 ~~~~~
1128 1214 DECR count FROM .ol-1 TO 0 DO chksum = .chksum + CH$RCHAR_A (ip);
1129 1215 neg_chksum = -.chksum;
1130 1216 ~~~~~
```

```

1131 1217 2 ! Get the stored checksum from the end of the record
1132 1218 2 !
1133 1219 char = CHSRCHAR_A (ip); ! Get the stored checksum
1134 1220 .new_start = .ip; ! Send back the start of the next record
1135 1221
1136 1222 IF .neg_chksum NEQ .char
1137 1223 THEN
1138 1224 BEGIN
1139 1225
1140 1226 ! The RSX/VMS utility FLX has been calculating incorrect checksums for records longer than 255 bytes. I
1141 1227 ! to include the high order byte of the length in the checksum. If the checksum is correct when we assu
1142 1228 ! that this has occurred, accept it as correct.
1143 1229
1144 1230 $debug_print_fao ('Record length !UL, checksum !OB, calc chksum !OB', .ol, .char, .neg_chksum);
1145 1231 chksum = .chksum - ((.ol+4) / 256); ! Pretend we never added the high byte
1146 1232 neg_chksum = -.chksum;
1147 1233 IF .neg_chksum NEQ .char
1148 1234 THEN
1149 1235 BEGIN
1150 1236 $debug_print_fao ('Record length !UL, checksum !OB, calc chksum !OB', .ol, .char, .neg_chksum);
1151 1237 RETURN findbin$k_chksum;
1152 1238 END;
1153 1239 END;
1154 1240
1155 1241 RETURN findbin$k_success;
1156 1242 END;

```

```

                                .EXTRN EXCH$UTIL_BLOCK_CHECK
                                .ENTRY PDP_FIND_BINARY_RECORD, Save R2,R3,R4,R5,- ; 1103
                                R6,R7,R8
                                MOVL FILB, R7 ; 1158
                                MOVL #56295674, R2
                                MOVZWL #495, R1
                                MOVL R7, R0
                                JSB EXCH$UTIL_BLOCK_CHECK
                                MOVL BUF_START, IP ; 1162
                                MOVL BUF_END, EOB ; 1163
                                MOVAB 4(R0), R1 ; 1172
                                CML R1, EOB
                                BGEQU 5$,
                                MOVB (IP)+, CHAR ; 1178
                                BEQL 1$, ; 1182
                                CMPB CHAR, #1 ; 1186
                                BNEQ 2$,
                                MOVZBL (IP)+, R1
                                BEQL 3$,
                                MOVL #4, R0 ; 1188
                                RET
                                MOVZWL (IP), OL ; 1192
                                SUBL2 #4, OL
                                MOVZBL (IP)+, R5 ; 1193
                                MOVZBL (IP)+, R6
                                MOVAB 1(R6)[R5], R8
                                MOVB R8, CHKSUM

```

```

                                01FC 0000
57 04 AC D0 00002
52 035B00FA 8F D0 00006
51 01EF 8F 3C 0000D
50 57 D0 00012
00000000G EF 16 00015
50 08 AC D0 0001B
52 0C AC D0 0001F
51 04 A0 9E 00023 1$:
52 51 D1 00027
3E 1E 0002A
54 80 90 0002C
F2 13 0002F
01 54 91 00031
05 12 00034
51 80 9A 00036
04 13 00039
50 04 D0 0003B 2$:
04 0003E
51 60 3C 0003F 3$:
51 04 C2 00042
55 80 9A 00045
56 80 9A 00048
58 01 A645 9E 0004B
53 58 90 00050

```

00000200	8F	51	D1	00053	CMPL	OL, #512	: 1197	
		04	1B	0005A	BLEQU	4\$: 1199	
	50	03	D0	0005C	MOVL	#3, R0	: 1203	
		04	0005F	RET			: 1205	
	56	01 A140	9E	00060	4\$: MOVAB	1(OL)[IP], R6	: 1209	
	52	56	D1	00065	CMPL	R6, EOB	: 1210	
		04	1F	00068	BLSSU	6\$: 1214	
	50	01	D0	0006A	5\$: MOVL	#1, R0	: 1215	
		04	0006D	RET			: 1219	
46	A7	50	D0	0006E	6\$: MOVL	IP, 70(R7)	: 1220	
42	A7	51	D0	00072	MOVL	OL, 66(R7)	: 1222	
	52	51	D0	00076	MOVL	OL, COUNT	: 1231	
		06	11	00079	BRB	8\$: 1232	
	55	80	9A	0007B	7\$: MOVZBL	(IP)+, R5	: 1233	
	53	55	80	0007E	ADDB2	R5, CHKSUM	: 1237	
	F7	52	F4	00081	8\$: SOBGEQ	COUNT, 7\$: 1241	
	52	53	8E	00084	MNEGB	CHKSUM, NEG_CHKSUM	: 1242	
	54	80	90	00087	MOVB	(IP)+, CHAR	: 1231	
10	BC	50	D0	0008A	MOVL	IP, @NEW START	: 1232	
	54	52	91	0008E	CMPB	NEG_CHKSUM, CHAR	: 1233	
		19	13	00091	BEQL	9\$: 1237	
	51	04	C0	00093	ADDL2	#4, R1	: 1241	
	51	00000100	8F	C6	00096	DIVL2	#256, R1	: 1242
	53	51	82	0009D	SUBB2	R1, CHKSUM	: 1232	
	52	53	8E	000A0	MNEGB	CHKSUM, NEG_CHKSUM	: 1233	
	54	52	91	000A3	CMPB	NEG_CHKSUM, CHAR	: 1237	
		04	13	000A6	BEQL	9\$: 1241	
	50	02	D0	000A8	MOVL	#2, R0	: 1242	
			04	000AB	RET		: 1241	
		50	D4	000AC	9\$: CLRL	R0	: 1242	
		04	000AE	RET			: 1242	

; Routine Size: 175 bytes, Routine Base: EXCH\$PDP_CODE + 051D

```

1158 1243 1 GLOBAL ROUTINE pdp_find_stream_record (filb : $ref_bblock, buf_start, %SBTTL 'pdp_find_stream_record'
1159 1244 1          buf_end : $ref_bvector, new_start) =
1160 1245 2 BEGIN
1161 1246 2 ++
1162 1247 2
1163 1248 2 FUNCTIONAL DESCRIPTION:
1164 1249 2
1165 1250 2     Scan buffer from start to end (if necessary) looking for a single stream record. The reformatted
1166 1251 2     record is copied to the record buffer in the filb. The address of the next unscanned byte is return
1167 1252 2
1168 1253 2 INPUTS:
1169 1254 2
1170 1255 2     buf_start - starting address in buffer to scan
1171 1256 2     buf_end   - one past the highest valid buffer address
1172 1257 2     filb     - pointer to the filb which contains the active record stream
1173 1258 2
1174 1259 2 IMPLICIT INPUTS:
1175 1260 2
1176 1261 2     none
1177 1262 2
1178 1263 2 OUTPUTS:
1179 1264 2
1180 1265 2     new_start - receives address of first unscanned byte
1181 1266 2
1182 1267 2 IMPLICIT OUTPUTS:
1183 1268 2
1184 1269 2     none
1185 1270 2
1186 1271 2 ROUTINE VALUE:
1187 1272 2
1188 1273 2     findstm$k_success - record placed in filb, all is well
1189 1274 2     ... k_ctrlz_eof   - ^Z at start of record
1190 1275 2     k_eob             - at end of buffer, no record found
1191 1276 2     k_no_term        - reached end of buffer in middle of record
1192 1277 2     k_bad_fmt        - record exceeds length of output buffer
1193 1278 2
1194 1279 2 SIDE EFFECTS:
1195 1280 2
1196 1281 2     none
1197 1282 2 --
1198 1283 2
1199 1284 2 $dbgtrc_prefix ('pdp_find_stream_record> ');
1200 1285 2
1201 1286 2 LOCAL
1202 1287 2     status
1203 1288 2     ;
1204 1289 2
1205 1290 2 REGISTER
1206 1291 2     ip,          ! Input pointer
1207 1292 2     op,          ! Output pointer
1208 1293 2     ol,          ! Output length
1209 1294 2     eob,        ! End of buffer
1210 1295 2     char        ! Current character
1211 1296 2     ;
1212 1297 2
1213 1298 2 $debug_print_lit ('entry');
1214 1299 2 $block_check(2, .filb, filb, 429);

```

```
1215 1300 2
1216 1301 2222 ! Set address of the filb record to the start of the filb record buffer
1217 1302 2222
1218 1303 2222 filb [filb$a_record] = filb [filb$t_record_buffer];
1219 1304 2222
1220 1305 2222 ! Initialize our local data segments
1221 1306 2222
1222 1307 2222 op = filb [filb$t_record_buffer]; ! Output pointer to the filb buffer
1223 1308 2222 ol = 0; ! Output length starts at zero
1224 1309 2222 ip = .buf_start; ! Input pointer at the start of the buffer
1225 1310 2222 eob = .buf_end; ! End of buffer pointer one past the end of the buffer
1226 1311 2222 status = findstm$k_success;
1227 1312 2222
1228 1313 2222 ! Start grabbing bytes
1229 1314 2222
1230 1315 2222 $debug_print_fao ('ip !XL, eob !XL, ol !XW, char '!AF'', .ip, .eob, .ol, 1, .ip);
1231 1316 2222 WHILE T
1232 1317 2222 DO
1233 1318 2222 BEGIN
1234 1319 2222
1235 1320 2222 ! Check for the end of either of the buffers
1236 1321 2222
1237 1322 2222 IF .ip GEQU .eob ! If the input pointer is past the end of the input buffer
1238 1323 2222 THEN
1239 1324 2222 BEGIN
1240 1325 2222 IF .ol EQL 0 ! If the output length is still zero
1241 1326 2222 THEN
1242 1327 2222 status = findstm$k_eob ! then end-of-buffer without any record
1243 1328 2222 ELSE
1244 1329 2222 status = findstm$k_no_term; ! otherwise record without terminator
1245 1330 2222 EXITLOOP;
1246 1331 2222 END;
1247 1332 2222
1248 1333 2222 IF .ol GTRU filb$s_record_buffer ! If the output length is gtr than the buffer (the buffer ac
1249 1334 2222 THEN ! has an extra guard byte at the end so no overrun problem)
1250 1335 2222 BEGIN
1251 1336 2222 status = findstm$k_bad_fmt; ! Our status is bad format record
1252 1337 2222 EXITLOOP;
1253 1338 2222 END;
1254 1339 2222
1255 1340 2222 ! Read the character and clear the high bit
1256 1341 2222
1257 1342 2222 char = CHSRCHAR A (ip); ! Read the new character and advance the input pointer
1258 1343 2222 char <7,1,0> = 0; ! Clear the high bit
1259 1344 2222
1260 1345 2222 ! Now look at the character and do something with it
1261 1346 2222
1262 1347 2222 SELECTONEU .char OF
1263 1348 2222 SET
1264 1349 2222
1265 1350 2222 [NUL, DEL, VT] :
1266 1351 2222 .
1267 1352 2222
1268 1353 2222 [CTRLZ] : ! Control/z marks end of file if the first char
1269 1354 2222 BEGIN
1270 1355 2222 IF .ol EQL 0
1271 1356 2222 THEN
```

```

: 1272      1357      S      BEGIN
: 1273      1358      S      status = findstm$sk_ctrlz_eof;      ! Fine, no record
: 1274      1359      S      EXITLOOP;
: 1275      1360      S      END
: 1276      1361      S      ELSE
: 1277      1362      S      BEGIN
: 1278      1363      S      CH$WCHAR_A (.char, op);
: 1279      1364      S      ol = .ol + 1;
: 1280      1365      S      END;
: 1281      1366      S      END;
: 1282      1367      S
: 1283      1368      S      [FF] :
: 1284      1369      S      BEGIN
: 1285      1370      S      CH$WCHAR_A (.char, op);
: 1286      1371      S      ol = .ol + 1;
: 1287      1372      S      EXITLOOP;
: 1288      1373      S      END;
: 1289      1374      S
: 1290      1375      S      [LF] :
: 1291      1376      S      BEGIN
: 1292      1377      S      IF .ol GTRU 0
: 1293      1378      S      THEN
: 1294      1379      S      BEGIN
: 1295      1380      S      IF CH$RCHAR (.op-1) EQL cr
: 1296      1381      S      THEN
: 1297      1382      S      ol = .ol - 1;
: 1298      1383      S      END;
: 1299      1384      S      EXITLOOP;
: 1300      1385      S      END;
: 1301      1386      S
: 1302      1387      S      [OTHERWISE] :
: 1303      1388      S      BEGIN
: 1304      1389      S      CH$WCHAR_A (.char, op);
: 1305      1390      S      ol = .ol + 1;
: 1306      1391      S      END;
: 1307      1392      S
: 1308      1393      S      TES;
: 1309      1394      S
: 1310      1395      S      END;
: 1311      1396      S
: 1312      1397      S      .new_start = .ip;
: 1313      1398      S      filb[filb$_record_len] = .ol;
: 1314      1399      S
: 1315      1400      S      $debug_print_fao ('recurd "'AF"', len !UL, status !UL', .ol, filb [filb$_record_buffer], .ol, .status);
: 1316      1401      S
: 1317      1402      S      RETURN .status;
: 1318      1403      S      END;

```

56	04	AC	D0	00002	.ENTRY	PDP_FIND_STREAM_RECORD, Save R2,R3,R4,R5,R6	; 1243
52	035B00FA	8F	D0	00006	MOVL	FILB, R6	; 1299
51	01AD	8F	3C	0000D	MOVL	#56295674, R2	
50		56	D0	00012	MOVZWL	#429, R1	
					MOVL	R6, R0	

		00000000G	EF	16	00015	JSB	EXCH\$UTIL_BLOCK_CHECK	...	1303
		015A	C6	9E	00018	MOVAB	346(R6), R0	...	1307
46	A6		50	D0	00020	MOVL	R0, 70(R6)	...	1308
	51		50	D0	00024	MOVL	R0, OP	...	1309
			52	D4	00027	CLRL	OL	...	1310
	50	08	AC	D0	00029	MOVL	BUF_START, IP	...	1311
	53	0C	AC	D0	0002D	MOVL	BUF_END, EOB	...	1322
			55	D4	00031	CLRL	STATUS	...	1325
	53		50	D1	00033	1\$:	C MPL IP, EOB	...	1327
			0E	1F	00036	BLSSU	3\$...	1329
			52	D5	00038	TSTL	OL	...	1333
			05	12	0003A	BNEQ	2\$...	1336
	55		02	D0	0003C	MOVL	#2, STATUS	...	1342
			5B	11	0003F	BRB	8\$...	1343
	55		03	D0	00041	2\$:	MOVL #3, STATUS	...	1350
			56	11	00044	BRB	8\$...	1353
00000200	8F		52	D1	00046	3\$:	C MPL OL, #512	...	1355
			05	1B	0004D	BLEQU	4\$...	1358
	55		04	D0	0004F	MOVL	#4, STATUS	...	1357
			48	11	00052	BRB	8\$...	1368
	54		80	90	00054	4\$:	MOVB (IP)+, CHAR	...	1370
	54	80	8F	8A	00057	BICB2	#128, CHAR	...	1371
			D6	13	0005B	BEQL	1\$...	1375
	0B		54	91	0005D	CMPB	CHAR, #11	...	1377
			D1	13	00060	BEQL	1\$...	1380
7F	8F		54	91	00062	CMPB	CHAR, #127	...	1382
			CB	13	00066	BEQL	1\$...	1389
	1A		54	91	00068	CMPB	CHAR, #26	...	1390
			09	12	0006B	BNEQ	5\$...	1397
			52	D5	0006D	TSTL	OL	...	1398
			24	12	0006F	BNEQ	7\$...	1402
	55		01	D0	00071	MOVL	#1, STATUS	...	1403
			26	11	00074	BRB	8\$...	
	0C		54	91	00076	5\$:	C MPB CHAR, #12	...	
			07	12	00079	BNEQ	6\$...	
	81		54	90	0007B	MOVB	CHAR, (OP)+	...	
			52	D6	0007E	INCL	OL	...	
			1A	11	0C080	BRB	8\$...	
	0A		54	91	00082	6\$:	C MPB CHAR, #10	...	
			0E	12	00085	BNEQ	7\$...	
			52	D5	00087	TSTL	OL	...	
			11	13	00089	BEQL	8\$...	
	0D	FF	A1	91	0008B	CMPB	-1(OP), #13	...	
			0B	12	0008F	BNEQ	8\$...	
			52	D7	0C091	DECL	OL	...	
			07	11	00093	BRB	8\$...	
	81		54	90	00095	7\$:	MOVB CHAR, (OP)+	...	
			52	D6	00098	INCL	OL	...	
			97	11	0009A	BRB	1\$...	
10	BC		50	D0	0009C	8\$:	MOVL IP, @NEW START	...	
42	A6		52	D0	000A0	MOVL	OL, 66(R6)	...	
	50		55	D0	000A4	MOVL	STATUS, R0	...	
			04	000A7	RET			...	

; Routine Size: 168 bytes, Routine Base: EXCH\$PDP_CODE + 05CC


```

1320 1404 1 GLOBAL ROUTINE exch$pdp_flush_write_buffer (ctx : $ref_bblock) = %SBTTL 'exch$pdp_flush_write_buffer
1321 1405 2 BEGIN
1322 1406 2 ++
1323 1407 2 FUNCTIONAL DESCRIPTION:
1324 1408 2
1325 1409 2 External entry to call buffer flush routine
1326 1410 2
1327 1411 2 INPUTS:
1328 1412 2
1329 1413 2 ctx - ctx pointer to context for an open RT11 file
1330 1414 2
1331 1415 2 IMPLICIT INPUTS:
1332 1416 2
1333 1417 2 none
1334 1418 2
1335 1419 2 OUTPUTS:
1336 1420 2
1337 1421 2 none
1338 1422 2
1339 1423 2 IMPLICIT OUTPUTS:
1340 1424 2
1341 1425 2 none
1342 1426 2
1343 1427 2 ROUTINE VALUE:
1344 1428 2
1345 1429 2 true if success, false if any error
1346 1430 2
1347 1431 2 SIDE EFFECTS:
1348 1432 2
1349 1433 2 error conditions will be signaled
1350 1434 2 --
1351 1435 2 $dbgtrc_prefix ('pdp_flush_write_buffer> ');
1352 1436 2
1353 1437 2 LOCAL
1354 1438 2 status
1355 1439 2 ;
1356 1440 2
1357 1441 2 $debug_print_lit ('entry');
1358 1442 2
1359 1443 2 $check_call (3, pdp_check_ctx, .ctx, 455); ! $block_check (2, .ctx, (dos11ctx or rt11ctx), 455)
1360 1444 2
1361 1445 2 ctx [ctx$flush] = true; ! Tells advance routine to flush the last block
1362 1446 2 status = pdp_buffer_advance_write (.ctx); ! Flush any blocks that are sitting in the output buffer
1363 1447 2 ctx [ctx$flush] = false; ! Clear the flush flag
1364 1448 2
1365 1449 2 RETURN .status;
1366 1450 1 END;

```

				0004	0000	.ENTRY	EXCH\$PDP_FLUSH_WRITE_BUFFER, Save R2	:	1404
			AC	DO	00002	MOVL	CTX, R2	:	1445
28	A2		04	88	00006	BISB2	#4, 40(R2)	:	
			52	DD	0000A	PUSHL	R2	:	1446
FAEA	CF		01	FB	0000C	CALLS	#1, PDP_BUFFER_ADVANCE_WRITE	:	

EXCHSPDP
V04-000

Small PDP-11 record structure routines
exchspdp_flush_write_buffer (ctx)

N 10
16-Sep-1984 01:11:46
14-Sep-1984 12:29:07

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

Page 40
(13)

28 A2 04 8A 00011
 04 00015

BICB2 #4, 40(R2)
RET

: 1447
: 1450

; Routine Size: 22 bytes, Routine Base: EXCHSPDP_CODE + 0674

EX
VO



```

: 1368 1451 1 GLOBAL ROUTINE exch$dpd_get (filb : $ref_bblock) = %SBTTL 'exch$dpd_get (filb)'
: 1369 1452 2 BEGIN
: 1370 1453 2 ++
: 1371 1454 2
: 1372 1455 2 FUNCTIONAL DESCRIPTION:
: 1373 1456 2
: 1374 1457 2 Common dispatch for RT11 get routines.
: 1375 1458 2
: 1376 1459 2 INPUTS:
: 1377 1460 2
: 1378 1461 2 filb - pointer to filb for an open RT11 file
: 1379 1462 2
: 1380 1463 2 IMPLICIT INPUTS:
: 1381 1464 2
: 1382 1465 2 none
: 1383 1466 2
: 1384 1467 2 OUTPUTS:
: 1385 1468 2
: 1386 1469 2 none
: 1387 1470 2
: 1388 1471 2 IMPLICIT OUTPUTS:
: 1389 1472 2
: 1390 1473 2 none
: 1391 1474 2
: 1392 1475 2 ROUTINE VALUE:
: 1393 1476 2
: 1394 1477 2 true if success, false if any error
: 1395 1478 2
: 1396 1479 2 SIDE EFFECTS:
: 1397 1480 2
: 1398 1481 2 error conditions will be signaled
: 1399 1482 2 --
: 1400 1483 2
: 1401 1484 2 $dbgtrc_prefix ('pdp_get> ');
: 1402 1485 2
: 1403 1486 2 LOCAL
: 1404 1487 2 buf_start, ! Pointer to next byte in the buffer
: 1405 1488 2 buf_end, ! -> one past the end of buffer
: 1406 1489 2 routn ! Address of action routine
: 1407 1490 2 ;
: 1408 1491 2
: 1409 1492 2 BIND
: 1410 1493 2 ctx = filb [filb$a_context] : $ref_bblock,
: 1411 1494 2 volb = filb [filb$a_assoc_volb] : $ref_bblock
: 1412 1495 2 ;

```

```

: 1414 1496 2 $debug_print_lit ('entry');
: 1415 1497 2
: 1416 1498 2 $block_check (2, .filb, filb, 456);
: 1417 1499 2 $block_check (2, .volb, volb, 493);
: 1418 1500 2 $check_call (1, pdp_check_ctx, .ctx, 494); ! $block_check (1, .ctx, (dos11ctx or rt11ctx), 494)
: 1419 1501 2 $logic_check (2, (.ctx [ctx$a_assoc_filb] EQL .filb), 134);
: 1420 1502 2 $logic_check (2, (.ctx [ctx$a_assoc_volb] EQL .volb), 135);
: 1421 1503 2 $logic_check (2, (IF .volb [volb$b_vol_format] EQL volb$k_vfmt_rt11 THEN (.ctx [ctx$l_cur_block] NEQ 0) ELSE
: 1422 1504 2
: 1423 1505 2 ! Get a pointer to the place to start scanning, and a pointer to the first byte past the end of the buffer
: 1424 1506 2
: 1425 1507 2 $logic_check (2, (.ctx [ctx$a_buffer] NEQ 0), 196);
: 1426 1508 2 buf_start = .ctx [ctx$a_buffer] + .ctx [ctx$l_cur_byte] +
: 1427 1509 2 ((.ctx [ctx$l_cur_block] - .ctx [ctx$l_buf_base_block]) * 512);
: 1428 1510 2 buf_end = .ctx [ctx$a_buffer] +
: 1429 1511 2 ((1 + .ctx [ctx$l_buf_high_block] - .ctx [ctx$l_buf_base_block]) * 512);
: 1430 1512 2
: 1431 1513 2 $$show_context;
: 1432 1514 2
: 1433 1515 2 ! Get the routine address for this specific record format
: 1434 1516 2
: 1435 1517 2 $trace_print_fao ('record format !UL', .filb [filb$b_rec_format]);
: 1436 1518 3 routn = (CASE .filb [filb$b_rec_format] FROM filb$k_rfmt_lobound TO filb$k_rfmt_hibound OF
: 1437 1519 3 SET
: 1438 1520 3 [filb$k_rfmt_binary] : pdp_get_binary;
: 1439 1521 3 [filb$k_rfmt_fixed] : pdp_get_fixed;
: 1440 1522 3 [filb$k_rfmt_stream] : pdp_get_stream;
: 1441 1523 3 [INRANGE] : $exch_signal_return (exch$_invrecfmt);
: 1442 1524 3 [filb$k_rfmt_invalid,
: 1443 1525 3 OUTRANGE] : BEGIN $logic_check (0, (false), 243); 0 END;
: 1444 1526 2 TES);
: 1445 1527 2
: 1446 1528 2 ! Now call the routine and return the status from it
: 1447 1529 2
: 1448 1530 2 RETURN jsb_get (.routn, .filb, .buf_start, .buf_end);
: 1449 1531 2
: 1450 1532 1 END;

```

					.EXTRN	EXCH\$_INVRECFMT	
					.ENTRY	EXCH\$PDP_GET, Save R2,R3,R4,R5,R6,R7,R8,R9,-;	1451
						R10	
					MOVAB	EXCH\$UTIL_BLOCK_CHECK, R10	
					MOVAB	LIB\$STOP, R9	
					MOVL	#EXCH\$BADLOGIC, R8	
					MOVL	FILB, R4	1493
					MOVL	#56295674, R2	1498
					MOVZWL	#456, R1	
					MOVL	R4, R0	
					JSB	EXCH\$UTIL_BLOCK_CHECK	
					MOVL	28(R4), R3	1499
					MOVL	#68878579, R2	
					MOVZWL	#493, R1	
					MOVL	R3, R0	
					JSB	EXCH\$UTIL_BLOCK_CHECK	

		7E	01EE	8F	3C	00041	MOVZWL	#494, -(SP)	1500
		52	20	A4	DD	00046	MOVL	32(R4), R2	
				52	DD	0004A	PUSHL	R2	
	00000000G	00		02	FB	0004C	CALLS	#2, PDP_CHECK_CTX	
		54	10	A2	D1	00053	CMPL	16(R2), -R4	1501
				0B	13	00057	BEQL	1\$	
		7E	86	8F	9A	00059	MOVZBL	#134, -(SP)	
				01	DD	0005D	PUSHL	#1	
				58	DD	0005F	PUSHL	R8	
		69		03	FB	00061	CALLS	#3, LIB\$STOP	
		53	14	A2	D1	00064	CMPL	20(R2), R3	1502
				0B	13	00068	BEQL	2\$	
		7E	87	8F	9A	0006A	MOVZBL	#135, -(SP)	
				01	DD	0006E	PUSHL	#1	
				58	DD	00070	PUSHL	R8	
		69		03	FB	00072	CALLS	#3, LIB\$STOP	
		03	58	A3	91	00075	CMPB	88(R3), #3	1503
				10	12	00079	BNEQ	3\$	
				1C	A2	D5	TSTL	28(R2)	
				0B	12	0007E	BNEQ	3\$	
		7E	B1	8F	9A	00080	MOVZBL	#177, -(SP)	
				01	DD	00084	PUSHL	#1	
				58	DD	00086	PUSHL	R8	
		69		03	FB	00088	CALLS	#3, LIB\$STOP	
		53	18	A2	D0	0008B	MOVL	24(R2), R3	1507
				0B	12	0008F	BNEQ	4\$	
		7E	C4	8F	9A	00091	MOVZBL	#196, -(SP)	
				01	DD	00095	PUSHL	#1	
				58	DD	00097	PUSHL	R8	
		69		03	FB	00099	CALLS	#3, LIB\$STOP	
		53	24	A2	C1	0009C	ADDL3	36(R2), R3, R1	1508
51				A2	C3	000A1	SUBL3	44(R2), 28(R2), R0	1509
50	1C			09	78	000A7	ASHL	#9, R0, R0	
50				50	C1	000AB	ADDL3	R0, R1, BUF_START	
56				A2	C3	000AF	SUBL3	44(R2), 48(R2), R2	1511
52	30			09	78	000B5	ASHL	#9, R2, R2	
52				57	C243	9E	MOVAB	512(R2)[R3], BUF_END	1510
				00	A4	8F	CASEB	40(R4), #0, #3	1518
0025	001E		0017	0008	000C4	5\$:	.WORD	6\$-5\$,- 7\$-5\$,- 8\$-5\$,- 9\$-5\$	
		7E	F3	8F	9A	000CC	MOVZBL	#243, -(SP)	1525
				01	DD	000D0	PUSHL	#1	
				58	DD	000D2	PUSHL	R8	
		69		03	FB	000D4	CALLS	#3, LIB\$STOP	
				50	D4	000D7	CLRL	ROUTN	
				13	11	000D9	BRB	10\$	
		50	0000V	CF	9E	000DB	MOVAB	PDP_GET_BINARY, ROUTN	1518
				0C	11	000E0	BRB	10\$	
		50	0000V	CF	9E	000E2	MOVAB	PDP_GET_FIXED, ROUTN	
				05	11	000E7	BRB	10\$	
		50	0000V	CF	9E	000E9	MOVAB	PDP_GET_STREAM, ROUTN	
		55		54	D0	000EE	MOVL	R4, R5	1530
				60	16	000F1	JSB	(ROUTN)	
				04	000F3		RET		1532

EXCHSPDP
V04-000

Small PDP-11 record structure routines
exch\$pdp_get (filb)

E 11
16-Sep-1984 01:11:46
14-Sep-1984 12:29:07

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

; Routine Size: 244 bytes, Routine Base: EXCHSPDP_CODE + 068A

```
1452 1533 1 GLOBAL ROUTINE pdp_get_binary (filb : $ref_bblock, %SBTTL 'pdp_get_binary (filb, buf_start, buf_end)'  
1453 1534 1 buf_start, buf_end) : jsb_get =  
1454 1535 2 BEGIN  
1455 1536 2 ++  
1456 1537 2  
1457 1538 2 FUNCTIONAL DESCRIPTION:  
1458 1539 2  
1459 1540 2 Return a pointer to the next formatted binary record in the file  
1460 1541 2  
1461 1542 2 INPUTS:  
1462 1543 2  
1463 1544 2 filb - pointer to filb for an open RT11 file  
1464 1545 2 buf_start - pointer to next byte in the buffer  
1465 1546 2 buf_end - pointer to one past the end of buffer  
1466 1547 2  
1467 1548 2 IMPLICIT INPUTS:  
1468 1549 2  
1469 1550 2 none  
1470 1551 2  
1471 1552 2 OUTPUTS:  
1472 1553 2  
1473 1554 2 none  
1474 1555 2  
1475 1556 2 IMPLICIT OUTPUTS:  
1476 1557 2  
1477 1558 2 none  
1478 1559 2  
1479 1560 2 ROUTINE VALUE:  
1480 1561 2  
1481 1562 2 true if success, false if any error  
1482 1563 2  
1483 1564 2 SIDE EFFECTS:  
1484 1565 2  
1485 1566 2 error conditions will be signaled  
1486 1567 2 --  
1487 1568 2  
1488 1569 2 $dbgtrc_prefix ('pdp_get_binary> ');  
1489 1570 2  
1490 1571 2 LOCAL  
1491 1572 2 new_start, ! Pointer to look next time.  
1492 1573 2 tmp,  
1493 1574 2 status  
1494 1575 2 ;  
1495 1576 2  
1496 1577 2 BIND  
1497 1578 2 ctx = filb [filb$a_context] : $ref_bblock,  
1498 1579 2 volb = filb [filb$a_assoc_volb] : $ref_bblock  
1499 1580 2 ;
```

```
1501 1581 2 $debug_print_lit ('entry');
1502 1582 2
1503 1583 2 ! Attempt to find a record in the current portion of the buffer
1504 1584 2
1505 1585 2 status = pdp_find_binary_record (.filb, .buf_start, .buf_end, new_start);
1506 1586 2
1507 1587 2 ! What did we see, what do we do
1508 1588 2
1509 1589 2 CASE .status FROM findbin$k_lobound TO findbin$k_hibound OF
1510 1590 2 SET
1511 1591 2
1512 1592 2 ! Success, update our next record pointer and return true
1513 1593 2
1514 1594 2 [findbin$k_success, findbin$k_chksum] :
1515 1595 2
1516 1596 2 BEGIN
1517 1597 2 IF .status EQL findbin$k_chksum
1518 1598 2 THEN
1519 1599 2   $exch_signal (exch$binchksum, 2, .filb [filb$l_result_name_len], filb [filb$t_result_na
1520 1600 2
1521 1601 2   tmp = .new_start - .ctx [ctx$a_buffer]; ! Save the updated position for the next get
1522 1602 2   ctx [ctx$l_cur_byte] = .tmp MOD 512;
1523 1603 2   ctx [ctx$l_cur_block] = (.tmp / 512) + .ctx [ctx$l_buf_base_block];
1524 1604 2   RETURN true; ! Found a record
1525 1605 2 END;
1526 1606 2
1527 1607 2 ! Hit the end of the buffer with no record, determine if EOF or need to read more buffer
1528 1608 2
1529 1609 2 [findbin$k_eob] :
1530 1610 2
1531 1611 2 BEGIN
1532 1612 2
1533 1613 2 $trace_print_lit ('findbin$k_eob status');
1534 1614 2 $$show_context;
1535 1615 2
1536 1616 2 ! If we are already at the eof block, then we have found EOF and can return
1537 1617 2
1538 1618 2 IF (.ctx [ctx$l_buf_high_block] GEQU .ctx [ctx$l_eof_block])
1539 1619 2 AND
1540 1620 2 (.ctx [ctx$l_eof_block] NEQ -1)
1541 1621 2 THEN
1542 1622 2   status = false
1543 1623 2
1544 1624 2 ! Otherwise, we can read in more data
1545 1625 2
1546 1626 2 ELSE
1547 1627 2 BEGIN
1548 1628 2 IF NOT (status = pdp_buffer_advance_read (.ctx))
1549 1629 2 THEN
1550 1630 2 BEGIN
1551 1631 2 IF .status EQL exch$stmrecfmt ! Means no room to read more blocks
1552 1632 2 THEN
1553 1633 2 BEGIN
1554 1634 2 status = exch$binrecfmt;
1555 1635 2 $exch_signal (.status, 2, .filb [filb$l_result_name_len], filb [filb$t_result_na
1556 1636 2 END
1557 1637 2 ELSE
```


	7E	F4	8F	9A	0001D	MOVZBL	#244, -(SP)	1663			
			01	DD	00021	PUSHL	#1				
	00000000G	00	00000000G	8F	DD	00023	PUSHL	#EXCH\$ BADLOGIC			
			03	FB	00029	CALLS	#3, LIB\$STOP				
			5E	11	00030	BRB	5\$				
		02	53	D1	00032	2\$:	CMPL	STATUS, #2			
			15	12	00035		BNEQ	3\$			
			5A	A5	9F	00037	PUSHAB	90(FILB)			
			3A	A5	DD	0003A	PUSHL	58(FILB)			
			02	DD	0003D	PUSHL	#2				
	00000000G	00	00000000G	8F	DD	0003F	PUSHL	#EXCH\$ BINCHKSUM			
			04	FB	00045	CALLS	#4, LIB\$SIGNAL				
			51	A5	DD	0004C	3\$:	MOVL	32(FILB), R1		
	50	6E	18	A1	C3	00050	SUBL3	24(R1), NEW_START, TMP	1601		
			50	01	7A	00055	EMUL	#1, TMP, #0, -(SP)	1602		
	7E	8E	00000200	8F	7B	0005A	EDIV	#512, (SP)+, R2, R2			
	52			52	DD	00063	MOVL	R2, 36(R1)			
				8F	C6	00067	DIVL2	#512, R0	1603		
	24	A1	00000200	B140	9E	0006E	MOVAB	@44(R1)[R0], 28(R1)			
				01	DD	00074	MOVL	#1, R0	1604		
				68	11	00077	BRB	13\$			
				50	DD	00079	4\$:	MOVL	32(FILB), R0		
	20	A0	30	A0	D1	0007D	CMPL	48(R0), 32(R0)	1618		
				0E	1F	00082	BLSSU	6\$			
	FFFFFFF	8F	20	A0	D1	00084	CMPL	32(R0), #-1	1620		
				04	13	0008C	BEQL	6\$			
				53	D4	0008E	CLRL	STATUS	1622		
				42	11	00090	5\$:	BRB	11\$		
				50	DD	00092	6\$:	PUSHL	R0		
	F7E9	CF		01	FB	00094	CALLS	#1, PDP_BUFFER_ADVANCE_READ	1628		
		53		50	DD	00099	MOVL	R0, STATUS			
		0B		53	E8	0009C	BLBS	STATUS, 7\$			
	00000000G	8F		53	D1	0009F	CMPL	STATUS, #EXCH\$ STMRECFMT	1631		
				0B	13	000A6	BEQL	8\$			
				34	11	000A8	BRB	12\$	1638		
				55	DD	000AA	7\$:	PUSHL	FILB		
	F5B	CF		01	FB	000AC	CALLS	#1, EXCHSPDP_GET	1641		
				2E	11	000B1	BRB	13\$			
				8F	DD	000B3	8\$:	MOVL	#EXCH\$ BINRECFMT, STATUS		
				07	11	000BA	BRB	10\$	1651		
				8F	DD	000BC	9\$:	MOVL	#EXCH\$ RECTOOBIG, STATUS		
				5A	A5	9F	000C3	10\$:	PUSHAB	90(FILB)	
				3A	A5	DD	000C6	PUSHL	58(FILB)	1658	
				02	DD	000C9	PUSHL	#2			
				53	DD	000CB	PUSHL	STATUS			
	00000000G	00		04	FB	000CD	CALLS	#4, LIB\$SIGNAL			
				50	A5	DD	000D4	11\$:	MOVL	32(FILB), R0	
				24	A0	D4	000D8	CLRL	36(R0)	1669	
				1C	A0	D4	000D9	CLRL	28(R0)	1670	
				50	53	DD	000DE	12\$:	MOVL	STATUS, R0	
				5E	04	C0	000E1	13\$:	ADDL2	#4, SP	1675
					05	000E4	RSB			1677	

; Routine Size: 229 bytes, Routine Base: EXCHSPDP_CODE + 077E

```
1599 1678 1 GLOBAL ROUTINE pdp_get_fixed (filb : $ref_bblock, %SBTTL 'pdp_get_fixed (filb, buf_start, buf_end)'  
1600 1679 1                                     buf_start, buf_end) : jsb_get =  
1601 1680 2 BEGIN  
1602 1681 2 ++  
1603 1682 2  
1604 1683 2 FUNCTIONAL DESCRIPTION:  
1605 1684 2  
1606 1685 2 Return a pointer to the next fixed-length record in the file  
1607 1686 2  
1608 1687 2 INPUTS:  
1609 1688 2  
1610 1689 2 filb - pointer to filb for an open RT11 file  
1611 1690 2 buf_start - pointer to next byte in the buffer  
1612 1691 2 buf_end - pointer to one past the end of buffer  
1613 1692 2  
1614 1693 2 IMPLICIT INPUTS:  
1615 1694 2  
1616 1695 2 none  
1617 1696 2  
1618 1697 2 OUTPUTS:  
1619 1698 2  
1620 1699 2 none  
1621 1700 2  
1622 1701 2 IMPLICIT OUTPUTS:  
1623 1702 2  
1624 1703 2 none  
1625 1704 2  
1626 1705 2 ROUTINE VALUE:  
1627 1706 2  
1628 1707 2 true if success, false if any error  
1629 1708 2  
1630 1709 2 SIDE EFFECTS:  
1631 1710 2  
1632 1711 2 error conditions will be signaled  
1633 1712 2 --  
1634 1713 2  
1635 1714 2 $dbgtrc_prefix ('pdp_get_fixed> ');  
1636 1715 2  
1637 1716 2 REGISTER  
1638 1717 2 five12,  
1639 1718 2 rec_size  
1640 1719 2 ;  
1641 1720 2  
1642 1721 2 LOCAL  
1643 1722 2 new_start, ! Pointer to look next time.  
1644 1723 2 tmp,  
1645 1724 2 status  
1646 1725 2 ;  
1647 1726 2  
1648 1727 2 BIND  
1649 1728 2 ctx = filb [filb$a_context] : $ref_bblock,  
1650 1729 2 volb = filb [filb$a_assoc_volb] : $ref_bblock  
1651 1730 2 ;  
1652 1731 2  
1653 1732 2 $debug_print_lit ('entry');  
1654 1733 2  
1655 1734 2 ! Preset some registers for a bit more speed
```

```

: 1656      1735 2  |
: 1657      1736 2  | five12 = 512;
: 1658      1737 2  | rec_size = .filb [filb$l_fixed_len];
: 1659      1738 2  |
: 1660      1739 2  | ! Get a pointer to the start of the next record
: 1661      1740 2  |
: 1662      1741 2  | new_start = .buf_start + .rec_size;
: 1663      1742 2  |
: 1664      1743 2  | ! See if the next record is in the buffer, EOF or advance the buffer if it isn't
: 1665      1744 2  |
: 1666      1745 2  | IF (.new_start - 1) GEQU .buf_end
: 1667      1746 2  | THEN
: 1668      1747 2  | BEGIN
: 1669      1748 2  |
: 1670      1749 2  |     ! If the EOF block is in the buffer
: 1671      1750 2  |
: 1672      1751 2  |     IF (.ctx [ctx$l_buf_high_block] GEQU .ctx [ctx$l_eof_block])
: 1673      1752 3  |         AND
: 1674      1753 4  |         (.ctx [ctx$l_eof_block] NEQ -1)
: 1675      1754 3  |     THEN
: 1676      1755 4  |         BEGIN
: 1677      1756 4  |
: 1678      1757 4  |             ! Set the next record position to invalid, and return false
: 1679      1758 4  |             !
: 1680      1759 4  |             ctx [ctx$l_cur_byte] = 0;
: 1681      1760 4  |             ctx [ctx$l_cur_block] = 0;
: 1682      1761 4  |             RETURN false;
: 1683      1762 4  |         END
: 1684      1763 4  |
: 1685      1764 4  |     ! Otherwise, read some more data in and recursively retry the get
: 1686      1765 4  |     !
: 1687      1766 3  |     ELSE
: 1688      1767 4  |         BEGIN
: 1689      1768 5  |             IF NOT (status = pdp_buffer_advance_read (.ctx))
: 1690      1769 4  |                 THEN
: 1691      1770 4  |                     RETURN .status;
: 1692      1771 4  |                 RETURN exch$pdp_get (.filb);           ! And then try it again
: 1693      1772 3  |             END;
: 1694      1773 2  |         END;
: 1695      1774 2  |
: 1696      1775 2  | $logic_check (2, ((.new_start - 1) LSSU .buf_end), 133);
: 1697      1776 2  |
: 1698      1777 2  | ! Use locate mode - point the filb record info at the buffer
: 1699      1778 2  | !
: 1700      1779 2  | filb [filb$a_record] = .buf_start;
: 1701      1780 2  | filb [filb$l_record_len] = .rec_size;
: 1702      1781 2  |
: 1703      1782 2  | ! Update the next record position
: 1704      1783 2  | !
: 1705      1784 2  | $logic_check (2, (.ctx [ctx$a_buffer] NEQ 0), 198);
: 1706      1785 2  | tmp = .new_start - .ctx [ctx$a_buffer]; ! Save the updated position for the next get
: 1707      1786 2  | ctx [ctx$l_cur_byte] = .tmp MOD .five12;
: 1708      1787 2  | ctx [ctx$l_cur_block] = (.tmp / .five12) + .ctx [ctx$l_buf_base_block];
: 1709      1788 2  |
: 1710      1789 2  | RETURN true;           ! Found a record
: 1711      1790 2  |
: 1712      1791 1  | END;
```

		52	0200	8F	3C	00000	PDP_GET_FIXED:.	MOVZWL	#512, FIVE12	1736
		53	35	A5	D0	00005		MOVL	53(FILB), REC_SIZE	1737
54		56		53	C1	00009		ADDL3	REC_SIZE, BUF_START, NEW_START	1741
				FF	A4	9F	00000	PUSHAB	-1(P4)	1745
		57		6E	D1	00010		CMPL	(SP), BUF_END	
				32	1F	00013		BLSSU	2\$	
	20	50	20	A5	D0	00015		MOVL	32(FILB), R0	1751
		A0	30	A0	D1	00019		CMPL	48(R0), 32(R0)	
				14	1F	0001E		BLSSU	1\$	
	FFFFFFF	8F	20	A0	D1	00020		CMPL	32(R0), #-1	1753
				0A	13	00C28		BEQL	1\$	
			24	A0	D4	0002A		CLRL	36(R0)	1759
			1C	A0	D4	0002D		CLRL	28(R0)	1760
				50	D4	00030		CLRL	R0	1767
				56	11	00032		BRB	4\$	
				50	DD	00034	1\$:	PUSHL	R0	1768
	F762	CF		01	FB	00036		CALLS	#1, PDP_BUFFER_ADVANCE_READ	
		4C		50	E9	0003B		BLBC	STATUS, -4\$	
				55	DD	0003E		PUSHL	FILB	1771
	FDE2	CF		01	FB	00040		CALLS	#1, EXCHSPDP_GET	
				43	11	00045		BRB	4\$	1767
	46	A5		56	D0	00047	2\$:	MOVL	BUF_START, 70(FILB)	1779
	42	A5		53	D0	0004B		MOVL	REC_SIZE, 66(FILB)	1780
		53	20	A5	D0	0004F		MOVL	32(FILB), R3	1784
			18	A3	D5	00053		TSTL	24(R3)	
				13	12	00056		BNEQ	3\$	
		7E	C6	8F	9A	00058		MOVZBL	#198, -(SP)	
				01	DD	0005C		PUSHL	#1	
			00000000G	8F	DD	0005E		PUSHL	#EXCHS_BADLOGIC	
	00000000G	00		03	FB	00064		CALLS	#3, LIB\$STOP	
		54	18	A3	C3	0006B	3\$:	SUBL3	24(R3), NEW_START, TMP	1785
7E	50	50		01	7A	00070		EMUL	#1, TMP, #0, -(SP)	1786
51	00	8E		52	7B	00075		EDIV	FIVE12, (SP)+, R1, R1	
		24		51	D0	0007A		MOVL	R1, 36(R3)	
		50		52	C6	0007E		DIVL2	FIVE12, R0	1787
		1C	A3	2C	B340	9E	00081	MOVAB	@44(R3)[R0], 28(R3)	
			50	01	D0	00087		MOVL	#1, R0	1789
			5E	04	C0	0008A	4\$:	ADDL2	#4, SP	1791
				05	0008D			RSB		

: Routine Size: 142 bytes, Routine Base: EXCHSPDP_CODE + 0863

```

: 1714 1792 1 GLOBAL ROUTINE pdp_get_stream (filb : $ref_bblock, %SBTTL 'pdp_get_stream (filb, buf_start, buf_end)'
: 1715 1793 1                                     buf_start, buf_end) : jsb_get =
: 1716 1794 2 BEGIN
: 1717 1795 2 ++
: 1718 1796 2
: 1719 1797 2 FUNCTIONAL DESCRIPTION:
: 1720 1798 2
: 1721 1799 2     Return a pointer to the next stream record in the file
: 1722 1800 2
: 1723 1801 2 INPUTS:
: 1724 1802 2
: 1725 1803 2     filb      - pointer to filb for an open RT11 file
: 1726 1804 2     buf_start - pointer to next byte in the buffer
: 1727 1805 2     buf_end  - pointer to one past the end of buffer
: 1728 1806 2
: 1729 1807 2 IMPLICIT INPUTS:
: 1730 1808 2
: 1731 1809 2     none
: 1732 1810 2
: 1733 1811 2 OUTPUTS:
: 1734 1812 2
: 1735 1813 2     none
: 1736 1814 2
: 1737 1815 2 IMPLICIT OUTPUTS:
: 1738 1816 2
: 1739 1817 2     none
: 1740 1818 2
: 1741 1819 2 ROUTINE VALUE:
: 1742 1820 2
: 1743 1821 2     true if success, false if any error
: 1744 1822 2
: 1745 1823 2 SIDE EFFECTS:
: 1746 1824 2
: 1747 1825 2     error conditions will be signaled
: 1748 1826 2 --
: 1749 1827 2
: 1750 1828 2 $dbgtrc_prefix ('pdp_get_stream> ');
: 1751 1829 2
: 1752 1830 2 LOCAL
: 1753 1831 2     new_start,           ! Pointer to look next time.
: 1754 1832 2     find_stat,
: 1755 1833 2     status
: 1756 1834 2     ;
: 1757 1835 2
: 1758 1836 2 BIND
: 1759 1837 2     ctx = filb [filb$a_context] : $ref_bblock,
: 1760 1838 2     volb = filb [filb$a_assoc_volb] : $ref_bblock
: 1761 1839 2     ;

```

```

: 1763      1840 2 $debug_print_lit ('entry');
: 1764      1841 2
: 1765      1842 2 : Attempt to find a record in this portion of the buffer
: 1766      1843 2
: 1767      1844 2 find_stat = pdp_find_stream_record (.filb, .buf_start, .buf_end, new_start);
: 1768      1845 2
: 1769      1846 2 : What did we see, what do we do
: 1770      1847 2
: 1771      1848 2 CASE .find_stat FROM findstm$k_lobound TO findstm$k_hibound OF
: 1772      1849 2 SET
: 1773      1850 2
: 1774      1851 2 : Success, update our next record pointer and return true
: 1775      1852 2
: 1776      1853 2 [findstm$k_success] :
: 1777      1854 2
: 1778      1855 2     BEGIN
: 1779      1856 2     LOCAL
: 1780      1857 2         tmp;
: 1781      1858 2     tmp = .new_start - .ctx [ctx$a_buffer]; ! Save the updated position for the next get
: 1782      1859 2     ctx [ctx$l_cur_byte] = .tmp MOD 512;
: 1783      1860 2     ctx [ctx$l_cur_block] = (.tmp / 512) + .ctx [ctx$l_buf_base_block];
: 1784      1861 2     RETURN true; ! Found a record
: 1785      1862 2     END;
: 1786      1863 2
: 1787      1864 2 : Found a control Z at the start of a record, done with this file
: 1788      1865 2
: 1789      1866 2 [findstm$k_ctrlz_eof] :
: 1790      1867 2
: 1791      1868 2     status = false;
: 1792      1869 2
: 1793      1870 2 : Hit the end of the buffer with no record, determine if EOF or need to read more buffer
: 1794      1871 2
: 1795      1872 2 [findstm$k_eob] :
: 1796      1873 2
: 1797      1874 2     BEGIN
: 1798      1875 2
: 1799      1876 2     $trace_print_lit ('findstm$k_eob status');
: 1800      1877 2     $$show_context;
: 1801      1878 2
: 1802      1879 2     ! If we are already at the eof block, then we have found EOF and can return
: 1803      1880 2
: 1804      1881 2     IF (.ctx [ctx$l_buf_high_block] GEQU .ctx [ctx$l_eof_block])
: 1805      1882 2     AND
: 1806      1883 2     (.ctx [ctx$l_eof_block] NEQ -1)
: 1807      1884 2     THEN
: 1808      1885 2         status = false
: 1809      1886 2
: 1810      1887 2     ! Otherwise, we can read in more data
: 1811      1888 2
: 1812      1889 2     ELSE
: 1813      1890 2     BEGIN
: 1814      1891 2     IF NOT (status = pdp_buffer_advance_read (.ctx))
: 1815      1892 2     THEN
: 1816      1893 2     BEGIN
: 1817      1894 2     IF .status EQL exch$_stmrecfmt ! Means no room to read more blocks
: 1818      1895 2     THEN
: 1819      1896 2     $exch_signal (.status, 2, .filb [filb$l_result_name_len], filb [filb$t_result_na

```

1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876

```

ELSE
RETURN .status;
END
ELSE
RETURN exch$pdp_get (.filb);
END;
END;
! Hit the end of the buffer with some record, determine if can read more buffer or final record is missi
[findstm$k_no_term] :
BEGIN
strace_print_lit ('findstm$k_no_term status');
$$show_context;
! If we are already at the eof block, then the record reaches to the end of the block
IF (.ctx [ctx$l_buf_high_block] GEQU .ctx [ctx$l_eof_block])
AND
(ctx [ctx$l_eof_block] NEQ -1)
THEN
BEGIN
LOCAL
tmp;
tmp = .new_start - .ctx [ctx$a_buffer]; ! Save the updated position for the next get
ctx [ctx$l_cur_byte] = .tmp MOD 512;
ctx [ctx$l_cur_block] = (.tmp / 512) + .ctx [ctx$l_buf_base_block];
RETURN true; ! Found a record
END
! Otherwise, we can read in more data
ELSE
BEGIN
IF NOT (status = pdp_buffer_advance_read (.ctx))
THEN
BEGIN
IF .status EQL exch$_stmrecfmt ! Means no room to read more blocks
THEN
sexch_signal (.status, 2, .filb [filb$l_result_name_len], filb [filb$t_result_na
ELSE
RETURN .status;
END
ELSE
RETURN exch$pdp_get (.filb);
END;
END;
! Found a badly formatted record
[findstm$k_bad_fmt] :
BEGIN

```



```
1877      1954      3      status = exch$stmrecfmt;  
1878      1955      3      sexch_signal (.status, 2, .filb [filb$l_result_name_len], filb [filb$t_result_name]);  
1879      1956      3      END;  
1880      1957      3      [INRANGE, OUTFRANGE] :  
1881      1958      3      $logic_check (0, (false), 245);  
1882      1959      3      TES;  
1883      1960      3      ! Set the next record position to invalid, and return false  
1884      1961      3      !  
1885      1962      3      ctx [ctx$l_cur_byte] = 0;  
1886      1963      3      ctx [ctx$l_cur_block] = 0;  
1887      1964      3      RETURN .status;  
1888      1965      3      !  
1889      1966      3      RETURN .status;  
1890      1967      3      !  
1891      1968      3      RETURN .status;  
1892      1969      3      !  
1893      1970      3      !  
1894      1971      3      END;
```

Address	OpCode	Operand 1	Operand 2	Operand 3	Operand 4	Instruction	Comment	Address	
		SE		04	C2 00000	PDP_GET_STREAM::			
			40E0	8F	BB 00003	SUBL2	#4, SP	1792	
				04	FB 00007	PUSHR	#M<R5,R6,R7,SP>	1844	
		FCCF	CF	50	CF 0000C	CALLS	#4, PDP_FIND_STREAM_RECORD		
			00			CASEL	FIND_STAT, #0, #4	1848	
0042	04	003A	001F	00A1		.WORD	2\$-1\$, 4\$-1\$, 3\$-1\$, 7\$-1\$, 12\$-1\$		
	7E		F5	8F	9A 0001A	MOVZBL	#245, -(SP)	1960	
				01	DD 0001E	PUSHL	#1		
		00000000G	00	8F	DD 00020	PUSHL	#EXCH\$ BADLOGIC		
				03	FB 00026	CALLS	#3, LIB\$STOP		
				1D	11 0002D	BRB	5\$		
	51		20	A5	DO 0002F	2\$:	MOVL	32(FILB), R1	1858
				32	11 00033	BRB	8\$		
		50	20	A5	DO 00035	3\$:	MOVL	32(FILB), R0	1881
	20	A0	30	A0	D1 00039		C MPL	48(R0), 32(R0)	
		FFFFFFF	8F	20	0E 1F 0003E		BLSSU	6\$	
				A0	D1 00040		C MPL	32(R0), #-1	1883
				04	13 00048		BEQL	6\$	
				53	D4 0004A	4\$:	CLRL	STATUS	1885
				7B	11 0004C	5\$:	BRB	14\$	
				50	DD 0004E	6\$:	PUSHL	R0	1891
				40	11 00050		BRB	10\$	
		51	20	A5	DO 00052	7\$:	MOVL	32(FILB), R1	1917
	20	A1	30	A1	D1 00056		C MPL	48(R1), 32(R1)	
		FFFFFFF	8F	20	33 1F 0005B		BLSSU	9\$	
				A1	D1 0005D		C MPL	32(R1), #-1	1919
				29	13 00065		BEQL	9\$	
		50	18	A1	C3 00067	8\$:	SUBL3	24(R1), NEW_START, TMP	1924
7E	00	50		01	7A 0006C		EMUL	#1, TMP, #0, -(SP)	
52	52	8E	00000200	8F	7B 00071		EDIV	#512, (SP)+, R2, R2	1925

24	A1		52	D0	0007A	MOVL	R2, 36(R1)		
	50	00000200	8F	C6	0007E	DIVL2	#512, R0	1926	
1C	A1		2C B140	9E	00085	MOVAB	244(R1)[R0], 28(R1)		
	50			01	D0	0008B	MOVL	#1, R0	1927
				46	11	0008E	BRB	16\$	
				51	DD	00090	9\$: PUSHL	R1	1934
F678	CF			01	FB	00092	10\$: CALLS	#1, PDP_BUFFER_ADVANCE_READ	
	53			50	DD	00097	MOVL	R0, STATUS	
	0B			53	EB	0009A	BLBS	STATUS, 11\$	
00000000G	8F			53	D1	0009D	CMPL	STATUS, #EXCH\$_STMRECFMT	1937
				12	13	000A4	BEQL	13\$	
				2B	11	000A6	BRB	15\$	1941
				55	DD	000A8	11\$: PUSHL	FILB	1944
FCEA	CF			01	FB	000AA	CALLS	#1, EXCH\$PDP_GET	
				25	11	000AF	BRB	16\$	
	53	00000000G		8F	D0	000B1	12\$: MOVL	#EXCH\$_STMRECFMT, STATUS	1954
			5A	A5	9F	000B8	13\$: PUSHAB	90(FILB)	1955
			3A	A5	DD	000BB	PUSHL	58(FILB)	
				02	DD	000BE	PUSHL	#2	
				53	DD	000C0	PUSHL	STATUS	
00000000G	00			04	FB	000C2	CALLS	#4, LIB\$SIGNAL	
	50		20	A5	D0	000C9	14\$: MOVL	32(FILB), R0	1966
			24	A0	D4	000CD	CLRL	36(R0)	
			1C	A0	D4	000D0	CLRL	28(R0)	1967
	50			53	D0	000D3	15\$: MOVL	STATUS, R0	1969
	5E			04	C0	000D6	16\$: ADDL2	#4, SP	1971
				05	000D9	RSB			

; Routine Size: 218 bytes, Routine Base: EXCH\$PDP_CODE + 08F1

```

1896 1972 1 GLOBAL ROUTINE exch$pdput = %SBTTL 'exch$pdput'
1897 1973 2 BEGIN
1898 1974 2 +-
1899 1975 2
1900 1976 2 FUNCTIONAL DESCRIPTION:
1901 1977 2
1902 1978 2 Common dispatch for RT11-style put routines. The main purpose of the extra dispatch is simplify the
1903 1979 2 mechanism for optimizing i/o transfers to physical mode when possible (for example RT11 -> RT11 does
1904 1980 2 need record mode).
1905 1981 2
1906 1982 2 INPUTS:
1907 1983 2
1908 1984 2 none
1909 1985 2
1910 1986 2 IMPLICIT INPUTS:
1911 1987 2
1912 1988 2 see the BIND expression
1913 1989 2
1914 1990 2 OUTPUTS:
1915 1991 2
1916 1992 2 none
1917 1993 2
1918 1994 2 IMPLICIT OUTPUTS:
1919 1995 2
1920 1996 2 see the BIND expression
1921 1997 2
1922 1998 2 ROUTINE VALUE:
1923 1999 2
1924 2000 2 value of format-specific put routine
1925 2001 2
1926 2002 2 SIDE EFFECTS:
1927 2003 2
1928 2004 2 none
1929 2005 2 --
1930 2006 2
1931 2007 2 $dbgtrc_prefix ('pdp_put> ');
1932 2008 2
1933 2009 2 LOCAL
1934 2010 2 buf_start,
1935 2011 2 buf_end,
1936 2012 2 routn
1937 2013 2 ;
1938 2014 2
1939 2015 2 BIND
1940 2016 2 copy = exch$a_gbl [excg$a_copy_work]: $ref_bblock, ! COPY verb work area
1941 2017 2 inp_filb = copy [copy$a_inp_filb] : $ref_bblock, ! pointer to the input filb with the record info
1942 2018 2 out_filb = copy [copy$a_out_filb] : $ref_bblock, ! pointer to filb for an open Files-11 output file
1943 2019 2 len = inp_filb [filb$l_record], ! length of the record
1944 2020 2 buf = inp_filb [filb$a_record], ! address of the record
1945 2021 2 ctx = out_filb [filb$a_context] : $ref_bblock, ! output file context block
1946 2022 2 volb = out_filb [filb$a_assoc_volb] : $ref_bblock ! output file volume block
1947 2023 2 ;
1948 2024 2
1949 2025 2 $debug_print_fao ('entry, format=!UL, len=!UL, buf[0:19]='!AF'', .out_filb [filb$b_rec_format], .len, 20, .b
1950 2026 2
1951 2027 2 $block_check (2, .inp_filb, filb, 466);
1952 2028 2 $block_check (2, .out_filb, filb, 467);

```

```

: 1953 2029 2 $check_call (1, pdp_check_ctx, .ctx, 537);          ! $block_check (1, .ctx, (dos11ctx or rt11ctx), 537)
: 1954 2030 2 $block_check (2, .volb, volb, 468);
: 1955 2031 2 $logic_check (2, (.ctx [ctx$a_assoc_filb] EQL .out_filb), 168);
: 1956 2032 2 $logic_check (2, (.ctx [ctx$a_assoc_volb] EQL .volb), 169);
: 1957 2033 2 $logic_check (2, (IF .volb [volb$b_vol_format] EQL volb$b_vfmt_rt11 THEN (.ctx [ctx$l_cur_block] NEQ 0) ELSE
: 1958 2034 2 $logic_check (2, (.len LEQU filb$s_record_buffer), 283);
: 1959 2035 2
: 1960 2036 2 ! Get pointers to the start of the next record position in the buffer, and to the end of the current buffer
: 1961 2037 2
: 1962 2038 2 $logic_check (2, (.ctx [ctx$a_buffer] NEQ 0), 200);
: 1963 2039 2 buf_start = .ctx [ctx$a_buffer] + .ctx [ctx$l_cur_byte] +
: 1964 2040 2 ((.ctx [ctx$l_cur_block] - .ctx [ctx$l_buf_base_block]) * 512);
: 1965 2041 2 buf_end = .ctx [ctx$a_buffer] +
: 1966 2042 2 ((1 + .ctx [ctx$l_buf_high_block] - .ctx [ctx$l_buf_base_block]) * 512);
: 1967 2043 2
: 1968 2044 2 ! Get the address of the record format specific routine
: 1969 2045 2
: 1970 2046 2 $trace_print_fao ('record format !UL', .out_filb [filb$b_rec_format]);
: 1971 2047 2 routn = (CASE .out_filb [filb$b_rec_format] FROM filb$k_rfmt_lobound TO filb$k_rfmt_hibound OF
: 1972 2048 2 SET
: 1973 2049 2     [filb$k_rfmt_binary] :      pdp_put_binary;
: 1974 2050 2     [filb$k_rfmt_fixed]  :      pdp_put_fixed;
: 1975 2051 2     [filb$k_rfmt_stream] :      pdp_put_stream;
: 1976 2052 2     [INRANGE]           :      $exch_signal_return (exch$_invrecfmt);
: 1977 2053 2     [filb$k_rfmt_invalid,
: 1978 2054 2     OUTRANGE]          :      BEGIN $logic_check (0, (false), 246); 0 END;
: 1979 2055 2 TES);
: 1980 2056 2
: 1981 2057 2 ! Now call that routine, returning the value of the routine
: 1982 2058 2
: 1983 2059 2 RETURN jsb_put (.routn, .buf_start, .buf_end, .ctx, .len, .buf);
: 1984 2060 1 END;

```

					.EXTRN	EXCH\$a_GBL	
			OFFC	00000	.ENTRY	EXCHSPDP_PUT, Save R2,R3,R4,R5,R6,R7,R8,R9,-;	1972
					MOVL	R10,R11	
					ADDL3	#EXCH\$a_BADLOGIC, R11	
					ADDL3	#4, EXCH\$a_GBL, R0	2016
					ADDL3	#60, (R0), R3	2017
					ADDL3	#68, (R0), R0	2018
					ADDL3	#66, (R3), R6	2019
					ADDL3	#70, (R3), R7	2020
					MOVL	(R0), R5	2021
					MOVL	#56295674, R2	2027
					MOVZWL	#466, R1	
					MOVL	(R3), R0	
					JSB	EXCH\$UTIL_BLOCK_CHECK	
					MOVL	#56295674, R2	2028
					MOVZWL	#467, R1	
					MOVL	R5, R0	
					JSB	EXCH\$UTIL_BLOCK_CHECK	
					MOVZWL	#537, -(SP)	2029
					MOVL	32(R5), R3	
					PUSHL	R3	

	00000000G	00		02	FB	00065		CALLS	#2, PDP_CHECK_CTX		
		54	1C	A5	DO	0006C		MOVL	28(R5), R4		2030
		52	041B00F3	8F	DO	0007U		MOVL	#68878579, R2		
		51	01D4	8F	3C	00077		MOVZWL	#468, R1		
		50		54	DO	0007C		MOVL	R4, R0		
			00000000G	EF	16	0007F		JSB	EXCH\$UTIL_BLOCK_CHECK		
		55	10	A3	D1	00085		CMPL	16(R3), R5		2031
				0F	13	00089		BEQL	1\$		
		7E	A8	8F	9A	0008B		MOVZBL	#168, -(SP)		
				01	DD	0008F		PUSHL	#1		
				5B	DD	00091		PUSHL	R11		
	00000000G	00		03	FB	00093		CALLS	#3, LIB\$STOP		
		54	14	A3	D1	0009A	1\$:	CMPL	20(R3), R4		2032
				0F	13	0009E		BEQL	2\$		
		7E	A9	8F	9A	000A0		MOVZBL	#169, -(SP)		
				01	DD	000A4		PUSHL	#1		
				5B	DD	000A6		PUSHL	R11		
	00000000G	00		03	FB	000A8		CALLS	#3, LIB\$STOP		
		03	58	A4	91	000AF	2\$:	CMPB	88(R4), #3		2033
				14	12	000B3		BNEQ	3\$		
				1C	A3	D5		TSTL	28(R3)		
				0F	12	000B8		BNEQ	3\$		
		7E	B0	8F	9A	000BA		MOVZBL	#176, -(SP)		
				01	DD	000BE		PUSHL	#1		
				5B	DD	000C0		PUSHL	R11		
	00000000G	00		03	FB	000C2		CALLS	#3, LIB\$STOP		
		00000200	8F	66	D1	000C9	3\$:	CMPL	(R6), #512		2034
				10	1B	000D0		BLEQU	4\$		
		7E	011B	8F	3C	000D2		MOVZWL	#283, -(SP)		
				01	DD	000D7		PUSHL	#1		
				5B	DD	000D9		PUSHL	R11		
	00000000G	00		03	FB	000DB		CALLS	#3, LIB\$STOP		
		52	18	A3	D0	000E2	4\$:	MOVL	24(R3), R2		2038
				0F	12	000E6		BNEQ	5\$		
		7E	C8	8F	9A	000E8		MOVZBL	#200, -(SP)		
				01	DD	000EC		PUSHL	#1		
				5B	DD	000EE		PUSHL	R11		
	00000000G	00		03	FB	000F0		CALLS	#3, LIB\$STOP		
		51	24	A3	C1	000F7	5\$:	ADDL3	36(R3), R2, R1		2039
		50	1C	A3	C3	000FC		SUBL3	44(R3), 28(R3), R0		2040
		50		09	78	00102		ASHL	#9, R0, R0		
		59		50	C1	00106		ADDL3	R0, R1, BUF_START		
		50	30	A3	C3	0010A		SUBL3	44(R3), 48(R3), R0		2042
		50		09	78	00110		ASHL	#9, R0, R0		
				5A	9E	00114		MOVAB	512(R0)[R2], BUF_END		2041
		03	00	042	A5	8F		CASEB	40(R5), #0, #3		2047
0029	0022		001B	0008		0011F	6\$:	.WORD	7\$-6\$, - 8\$-6\$, - 9\$-6\$, - 10\$-6\$, -		
		7E	F6	8F	9A	00127	7\$:	MOVZBL	#246, -(SP)		2054
				01	DD	0012B		PUSHL	#1		
				5B	DD	0012D		PUSHL	R11		
	00000000G	00		03	FB	0012F		CALLS	#3, LIB\$STOP		
				50	D4	00136		CLRL	ROUTN		
				13	11	00138		BRB	11\$		
		50	0000V	CF	9E	0013A	8\$:	MOVAB	PDP_PUT_BINARY, ROUTN		2047

EXCH\$PDP
V04-000

Small PDP-11 record structure routines
exch\$pdput

H 12
16-Sep-1984 01:11:46
14-Sep-1984 12:29:07

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

Page 60
(21)

50	0000V	0C	11	0013F	9\$:	BRB	11\$:
		CF	9E	00141		MOVAB	PDP_PUT_FIXED, ROUTN	:
50	0000V	05	11	00146	10\$:	BRB	11\$:
		CF	9E	00148		MOVAB	PDP_PUT_STREAM, ROUTN	:
		67	DD	0014D	11\$:	PUSHL	(R7)	2059
		66	DD	0014F		PUSHL	(R6)	:
		53	DD	00151		PUSHL	R3	:
		60	16	00153		JSB	(ROUTN)	:
		04	04	00155		RET		2060

: Routine Size: 342 bytes, Routine Base: EXCH\$PDP_CODE + 09CB

```

: 1986 2061 1 GLOBAL ROUTINE pdp_put_binary (buf_start, buf_end, ctx : $ref_bblock, len, buf) : jsb_put = %SBTTL 'pdp_
: 1987 2062 2 BEGIN
: 1988 2063 3 +-
: 1989 2064 4
: 1990 2065 5 FUNCTIONAL DESCRIPTION:
: 1991 2066 6
: 1992 2067 7 Add the next formatted binary record in the file
: 1993 2068 8
: 1994 2069 9 INPUTS:
: 1995 2070 10
: 1996 2071 11 buf_start - Pointer to next byte in the buffer
: 1997 2072 12 buf_end - Pointer to one past the end of buffer
: 1998 2073 13 ctx - Output file context block
: 1999 2074 14 len - Length of the record to be put
: 2000 2075 15 buf - Address of the record
: 2001 2076 16
: 2002 2077 17 IMPLICIT INPUTS:
: 2003 2078 18
: 2004 2079 19 see the BIND expression
: 2005 2080 20
: 2006 2081 21 OUTPUTS:
: 2007 2082 22
: 2008 2083 23 none
: 2009 2084 24
: 2010 2085 25 IMPLICIT OUTPUTS:
: 2011 2086 26
: 2012 2087 27 see the BIND expression
: 2013 2088 28
: 2014 2089 29 ROUTINE VALUE:
: 2015 2090 30
: 2016 2091 31 true if success, false if any error
: 2017 2092 32
: 2018 2093 33 SIDE EFFECTS:
: 2019 2094 34
: 2020 2095 35 error conditions will be signaled
: 2021 2096 36 --
: 2022 2097 37
: 2023 2098 38 $dbgtrc_prefix ('pdp_put_binary> ');
: 2024 2099 39
: 2025 2100 40 REGISTER
: 2026 2101 41 next_rec,
: 2027 2102 42 tmp
: 2028 2103 43 :
: 2029 2104 44
: 2030 2105 45 BIND
: 2031 2106 46 copy = exch$a_gbl [excg$a_copy_work]: $ref_bblock, ! COPY verb work area
: 2032 2107 47 out_filb = copy [copy$a_out_fi[b] : $ref_bblock ! pointer to filb for an open Files-11 output file
: 2033 2108 48 :
: 2034 2109 49
: 2035 2110 50 $debug_print_fao ('entry, len=!UL, buf[0:19]='!AF'', .len, 20, .buf);
: 2036 2111 51
: 2037 2112 52 ! Get a pointer to the start of the next record after this one
: 2038 2113 53 !
: 2039 2114 54 next_rec = .buf_start + .len + 5; ! <sentinel-word> <length-word> <record-data> <checksum-byte>
: 2040 2115 55 !
: 2041 2116 56 ! See if the next record will fit in the buffer, EOF or advance the buffer if it isn't
: 2042 2117 57 !

```

```

: 2043      2118 2 IF (.next_rec - 1) GEQU .buf_end
: 2044      2119 ~~~~~ THEN
: 2045      2120 ~~~~~ RETURN pdp_buffer_check (.ctx, .out_filb);
: 2046      2121 ~~~~~
: 2047      2122 ~~~~~ ! Move the record to the buffer
: 2048      2123 ~~~~~ !
: 2049      2124 ~~~~~ pdp_copy_binary_record (.len, .buf, .buf_start);
: 2050      2125 ~~~~~
: 2051      2126 ~~~~~ ! Update the next record position and return
: 2052      2127 ~~~~~ !
: 2053      2128 ~~~~~ RETURN pdp_buffer_update (.ctx, .next_rec);
: 2054      2129 ~~~~~
: 2055      2130 1 END:

```

	55		59	D0 00000	PDP_PUT_BINARY::		
					MOVL	R9, R5	: 2061
50 00000000G	EF		04	C1 00003	ADDL3	#4, EXCH\$A_GBL, R0	: 2106
50	60	00000044	8F	C1 0000B	ADDL3	#68, (R0), R0	: 2107
59	55	08	AE	C1 00013	ADDL3	LEN, BUF_START, R9	: 2114
	54	05	A9	9E 00018	MOVAB	5(R9), NEXT_REC	
	59	FF	A4	9E 0001C	MOVAB	-1(R4), R9	: 2118
	5A		59	D1 00020	CMPL	R9, BUF_END	
			0A	1F 00023	BLSSU	1\$	
	53		60	D0 00025	MOVL	(R0), R3	: 2120
	52	04	AE	D0 00028	MOVL	CTX, R2	
			F7B3	31 0002C	BRW	PDP_BUFFER_CHECK	
			55	DD 0002F	PUSHL	BUF_START	: 2124
			10	AE DD 00031	PUSHL	BUF	
			10	AE DD 00034	PUSHL	LEN	
F902	CF		03	FB 00037	CALLS	#3, PDP_COPY_BINARY_RECORD	
	53		54	D0 0003C	MOVL	NEXT_REC, R3	: 2128
	52	04	AE	D0 0003F	MOVL	CTX, R2	
			F7CF	31 00043	BRW	PDP_BUFFER_UPDATE	

; Routine Size: 70 bytes, Routine Base: EXCH\$PDP_CODE + 0B21


```
2057 2131 1 GLOBAL ROUTINE pdp_put_fixed (buf_start, buf_end, ctx : $ref_bblock, len, buf) : jsb_put = %SBTTL 'pdp_
2058 2132 2 BEGIN
2059 2133 3 ++
2060 2134 4
2061 2135 5 FUNCTIONAL DESCRIPTION:
2062 2136 6
2063 2137 7 INPUTS:
2064 2138 8
2065 2139 9     buf_start - Pointer to next byte in the buffer
2066 2140 10     buf_end   - Pointer to one past the end of buffer
2067 2141 11     ctx      - Output file context block
2068 2142 12     len      - Length of the record to be put
2069 2143 13     buf      - Address of the record
2070 2144 14
2071 2145 15 IMPLICIT INPUTS:
2072 2146 16
2073 2147 17     see the BIND expression
2074 2148 18
2075 2149 19 OUTPUTS:
2076 2150 20
2077 2151 21     none
2078 2152 22
2079 2153 23 IMPLICIT OUTPUTS:
2080 2154 24
2081 2155 25     see the BIND expression
2082 2156 26
2083 2157 27 ROUTINE VALUE:
2084 2158 28
2085 2159 29     true if success, false if any error
2086 2160 30
2087 2161 31 SIDE EFFECTS:
2088 2162 32
2089 2163 33     error conditions will be signaled
2090 2164 34 --
2091 2165 35
2092 2166 36 $dbgtrc_prefix ('pdp_put_fixed> ');
2093 2167 37
2094 2168 38 REGISTER
2095 2169 39     rec_size,
2096 2170 40     next_rec,           ! Pointer to look next time.
2097 2171 41     tmp
2098 2172 42     ;
2099 2173 43
2100 2174 44 BIND
2101 2175 45     copy = exch$a_gbl [excg$a_copy_work]: $ref_bblock, ! COPY verb work area
2102 2176 46     out_filb = copy [copy$a_out_fi[b] : $ref_bblock ! pointer to filb for an open Files-11 output file
2103 2177 47     ;
2104 2178 48
2105 2179 49 $debug_print_fao ('entry, len=!UL, buf[0:19]='!AF'', .len, 20, .buf);
2106 2180 50
2107 2181 51 rec_size = .out_filb [filb$l_fixed_len];
2108 2182 52
2109 2183 53 ! Get a pointer to the start of the next record after this one
2110 2184 54 !
2111 2185 55 next_rec = .buf_start + .rec_size;
2112 2186 56
2113 2187 57 ! See if the next record will fit in the buffer, EOF or advance the buffer if it isn't
```

```

: 2114      2188 2 !
: 2115      2189 2 IF (.next_rec - 1) GEQU .buf_end
: 2116      2190 2 THEN
: 2117      2191 2     RETURN pdp_buffer_check (.ctx, .out_filb);
: 2118      2192 2
: 2119      2193 2 ! Move the record to the buffer
: 2120      2194 2
: 2121      2195 2 CH$COPY (.len, .buf, .out_filb [filb$b_pad_char], .rec_size, .buf_start);
: 2122      2196 2
: 2123      2197 2 ! Update the next record position and return
: 2124      2198 2
: 2125      2199 2 RETURN pdp_buffer_update (.ctx, .next_rec);
: 2126      2200 2
: 2127      2201 1 END;

```

50	00000000G	EF	04	C1	00000	PDP_PUT_FIXED::		
						ADDL3	#4, EXCH\$A_GBL, R0	: 2175
50	60	00000044	8F	C1	00008	ADDL3	#68, (R0), R0	: 2176
	55		60	D0	00010	MOVL	(R0), R5	: 2181
	54	35	A5	D0	00013	MOVL	53(R5), REC_SIZE	
56	59		54	C1	00017	ADDL3	REC_SIZE, BUF_START, NEXT_REC	: 2185
	53	FF	A6	9E	0001B	MOVAB	-1(R6), R3	: 2189
	5A		53	D1	0001F	CML	R3, BUF_END	
			0A	1F	00022	BLSSU	1\$	
	53		55	D0	00024	MOVL	R5, R3	: 2191
	52	04	AE	D0	00027	MOVL	CTX, R2	
54	39	A5	OC	BE	08	AE	2C 00C2E 1\$:	: 2195
					69		00036	
	53		56	D0	00037	MOVL	NEXT_REC, R3	: 2199
	52	04	AE	D0	0003A	MOVL	CTX, R2	
			F78E	31	0003E	BRW	PDP_BUFFER_UPDATE	

: Routine Size: 65 bytes, Routine Base: EXCH\$PDP_CODE + 0B67

```
2129 2202 1 GLOBAL ROUTINE pdp_put_stream (buf_start, buf_end, ctx : $ref_bblock, len, buf) : jsb_put = %SBTTL 'pdp_
2130 2203 2 BEGIN
2131 2204 3 ++
2132 2205 4
2133 2206 5 FUNCTIONAL DESCRIPTION:
2134 2207 6
2135 2208 7     Add the next stream record in the file
2136 2209 8
2137 2210 9 INPUTS:
2138 2211 10
2139 2212 11     buf_start - Pointer to next byte in the buffer
2140 2213 12     buf_end   - Pointer to one past the end of buffer
2141 2214 13     ctx      - Output file context block
2142 2215 14     len     - Length of the record to be put
2143 2216 15     buf     - Address of the record
2144 2217 16
2145 2218 17 IMPLICIT INPUTS:
2146 2219 18
2147 2220 19     see the BIND expression
2148 2221 20
2149 2222 21 OUTPUTS:
2150 2223 22
2151 2224 23     none
2152 2225 24
2153 2226 25 IMPLICIT OUTPUTS:
2154 2227 26
2155 2228 27     see the BIND expression
2156 2229 28
2157 2230 29 ROUTINE VALUE:
2158 2231 30
2159 2232 31     true if success, false if any error
2160 2233 32
2161 2234 33 SIDE EFFECTS:
2162 2235 34
2163 2236 35     error conditions will be signaled
2164 2237 36 --
2165 2238 37
2166 2239 38 $dbgtrc_prefix ('pdp_put_stream> ');
2167 2240 39
2168 2241 40 REGISTER
2169 2242 41     actual_len,
2170 2243 42     next_rec,
2171 2244 43     tmp
2172 2245 44     :
2173 2246 45
2174 2247 46 BIND
2175 2248 47     copy = exch$a_gbl [excg$a_copy_work]: $ref_bblock, ! COPY verb work area
2176 2249 48     out_filb = copy [copy$a_out_fi[b] : $ref_bblock ! pointer to filb for an open Files-11 output file
2177 2250 49     :
2178 2251 50
2179 2252 51 $debug_print_fao ('entry, len=!UL, buf[0:19]='!AF'', .len, 20, .buf);
2180 2253 52
2181 2254 53 ! Get a pointer to the start of the next record after this one
2182 2255 54 !
2183 2256 55 next_rec = .buf_start + .len + 2; ! Assume record plus <CR><LF>
2184 2257 56
2185 2258 57 ! See if the next record will fit in the buffer, EOF or advance the buffer if it isn't
```

```

: 2186      2259 2 !
: 2187      2260 2 IF (.next_rec - 1) GEQU .buf_end
: 2188      2261 2 THEN
: 2189      2262 2     RETURN pdp_buffer_check (.ctx, .out_filb);
: 2190      2263 2
: 2191      2264 2 ! Move the record to the buffer
: 2192      2265 2
: 2193      2266 2 actual_len = pdp_copy_stream_record (.len, .buf, .buf_start);
: 2194      2267 2
: 2195      2268 2 ! Update the next record position and return
: 2196      2269 2
: 2197      2270 2 RETURN pdp_buffer_update (.ctx, .buf_start + .actual_len);
: 2198      2271 2
: 2199      2272 1 END;

```

	54		59	D0 00000	PDP_PUT_STREAM::		
					MOVL	R9, R4	: 2202
50	00000000G	EF	04	C1 00003	ADDL3	#4, EXCHSA_GBL, R0	: 2248
51		60	00000044	8F C1 0000B	ADDL3	#68, (R0), R1	: 2249
59		54	08	AE C1 00013	ADDL3	LEN, BUF_START, R9	: 2256
		50	02	A9 9E 00018	MOVAB	2(R9), NEXT_REC	
				50 D7 0001C	DECL	R0	: 2260
		5A		50 D1 0001E	CML	R0, BUF_END	
				0A 1F 00021	BLSSU	1\$	
		53		61 D0 00023	MOVL	(R1), R3	: 2262
		52	04	AE D0 00026	MOVL	CTX, R2	
				F72E 31 0002A	BRW	PDP_BUFFER_CHECK	
				54 DD 0002D	1\$: PUSHL	BUF_START	: 2266
			10	AE DD 0002F	PUSHL	BUF	
			10	AE DD 00032	PUSHL	LEN	
	F8BD	CF		03 FB 00035	CALLS	#3, PDP_COPY_STREAM_RECORD	
53		54		50 C1 0003A	ADDL3	ACTUAL_LEN, BUF_START, R3	: 2270
		52	04	AE D0 0003E	MOVL	CTX, R2	
				F749 31 00042	BRW	PDP_BUFFER_UPDATE	: 2270

; Routine Size: 69 bytes, Routine Base: EXCH\$PDP_CODE + 0B85

EXCH\$PDP
V04-000

Small PDP-11 record structure routines
pdp_put_stream

B 13
16-Sep-1984 01:11:46
14-Sep-1984 12:29:07

VAX-11 Bliss-32 V4.0-742
[EXCHNG.SRC?EXCPDP.B32;1

Page 67
(25)

: 2201
: 2202
2273 1 END
2274 0 ELUDOM

.EXTRN LIB\$SIGNAL, LIB\$STOP

PSECT SUMMARY

Name	Bytes	Attributes
EXCH\$PDP_CODE	3053	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	3	0	1000	00:01.8
\$_\$255\$DUA28:[EXCHNG.OBJ]EXCLIB.L32;1	1151	99	8	79	00:01.3

COMMAND QUALIFIERS

BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS\$:EXCPDP/OBJ=OBJ\$:EXCPDP MSRC\$:EXCPDP/UPDATE=(ENH\$:EXCPDP)

: Size: 3053 code + 0 data bytes
: Run Time: 00:57.4
: Elapsed Time: 02:38.6
: Lines/CPU Min: 2377
: Lexemes/CPU-Min: 21756
: Memory Used. 187 pages
: Compilation Complete

Terminal window 1	Terminal window 2	Terminal window 3	Terminal window 4	Terminal window 5	Terminal window 6	Terminal window 7	Terminal window 8	Terminal window 9	Terminal window 10	Terminal window 11	Terminal window 12	Terminal window 13	Terminal window 14	Terminal window 15	Terminal window 16
Terminal window 17	Terminal window 18	Terminal window 19	Terminal window 20	Terminal window 21	Terminal window 22	Terminal window 23	Terminal window 24	Terminal window 25	Terminal window 26	Terminal window 27	Terminal window 28	Terminal window 29	Terminal window 30	Terminal window 31	Terminal window 32
Terminal window 33	Terminal window 34	Terminal window 35	Terminal window 36	Terminal window 37	Terminal window 38	Terminal window 39	Terminal window 40	Terminal window 41	Terminal window 42	Terminal window 43	Terminal window 44	Terminal window 45	Terminal window 46	Terminal window 47	Terminal window 48
Terminal window 49	Terminal window 50	Terminal window 51	Terminal window 52	Terminal window 53	Terminal window 54	Terminal window 55	Terminal window 56	Terminal window 57	Terminal window 58	Terminal window 59	Terminal window 60	Terminal window 61	Terminal window 62	Terminal window 63	Terminal window 64
Terminal window 65	Terminal window 66	Terminal window 67	Terminal window 68	Terminal window 69	Terminal window 70	Terminal window 71	Terminal window 72	Terminal window 73	Terminal window 74	Terminal window 75	Terminal window 76	Terminal window 77	Terminal window 78	Terminal window 79	Terminal window 80
Terminal window 81	Terminal window 82	Terminal window 83	Terminal window 84	Terminal window 85	Terminal window 86	Terminal window 87	Terminal window 88	Terminal window 89	Terminal window 90	Terminal window 91	Terminal window 92	Terminal window 93	Terminal window 94	Terminal window 95	Terminal window 96
Terminal window 97	Terminal window 98	Terminal window 99	Terminal window 100	Terminal window 101	Terminal window 102	Terminal window 103	Terminal window 104	Terminal window 105	Terminal window 106	Terminal window 107	Terminal window 108	Terminal window 109	Terminal window 110	Terminal window 111	Terminal window 112
Terminal window 113	Terminal window 114	Terminal window 115	Terminal window 116	Terminal window 117	Terminal window 118	Terminal window 119	Terminal window 120	Terminal window 121	Terminal window 122	Terminal window 123	Terminal window 124	Terminal window 125	Terminal window 126	Terminal window 127	Terminal window 128
Terminal window 129	Terminal window 130	Terminal window 131	Terminal window 132	Terminal window 133	Terminal window 134	Terminal window 135	Terminal window 136	Terminal window 137	Terminal window 138	Terminal window 139	Terminal window 140	Terminal window 141	Terminal window 142	Terminal window 143	Terminal window 144
Terminal window 145	Terminal window 146	Terminal window 147	Terminal window 148	Terminal window 149	Terminal window 150	Terminal window 151	Terminal window 152	Terminal window 153	Terminal window 154	Terminal window 155	Terminal window 156	Terminal window 157	Terminal window 158	Terminal window 159	Terminal window 160

EXCMIN
LIS

EXMSG
LIS

EXCPDP
LIS

EXCMOUN
LIS

EXCRT11
LIS