

FFFFFFFFFFFFFFFF	111	111	AAAAAAAAA	
FFFFFFFFFFFFFFFF	111	111	AAAAAAAAA	
FFFFFFFFFFFFFFFF	111	111	AAAAAAAAA	
FFF	111111	111111	AAA	AAA
FFF	111111	111111	AAA	AAA
FFF	111111	111111	AAA	AAA
FFF	111	111	AAA	AAA
FFF	111	111	AAA	AAA
FFF	111	111	AAA	AAA
FFFFFFFFFFFFFF	111	111	AAA	AAA
FFFFFFFFFFFFFF	111	111	AAA	AAA
FFFFFFFFFFFFFF	111	111	AAA	AAA
FFF	111	111	AAAAAAAAAAAAAAAA	
FFF	111	111	AAAAAAAAAAAAAAAA	
FFF	111	111	AAAAAAAAAAAAAAAA	
FFF	111	111	AAA	AAA
FFF	111	111	AAA	AAA
FFF	111	111	AAA	AAA
FFF	111	111	AAA	AAA
FFF	111111111	111111111	AAA	AAA
FFF	111111111	111111111	AAA	AAA
FFF	111111111	111111111	AAA	AAA

```

LL               IIIIII               SSSSSSSS
LL               IIIIII               SSSSSSSS
LL               II                 SS
LL               II                 SS
LL               II                 SS
LL               II                 SS
LL               II                 SSSSSS
LL               II                 SSSSSS
LL               II                 SS
LL               II                 SS
LL               II                 SS
LL               II                 SS
LLLLLLLLLLLLLL  IIIIII               SSSSSSSS
LLLLLLLLLLLLLL  IIIIII               SSSSSSSS

```



```
1 0001 0 MODULE CREHDR (  
2 0002 0 LANGUAGE (BLISS32),  
3 0003 0 IDENT = 'V04-000'  
4 0004 0 ) =  
5 0005 1 BEGIN  
6 0006 1  
7 0007 1  
8 0008 1 *****  
9 0009 1 *  
10 0010 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY *  
11 0011 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS. *  
12 0012 1 * ALL RIGHTS RESERVED. *  
13 0013 1 *  
14 0014 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED *  
15 0015 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE *  
16 0016 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER *  
17 0017 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY *  
18 0018 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY *  
19 0019 1 * TRANSFERRED. *  
20 0020 1 *  
21 0021 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE *  
22 0022 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT *  
23 0023 1 * CORPORATION. *  
24 0024 1 *  
25 0025 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS *  
26 0026 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL. *  
27 0027 1 *  
28 0028 1 *  
29 0029 1 *****  
30 0030 1  
31 0031 1 ++  
32 0032 1  
33 0033 1 FACILITY: F11ACP Structure Level 1  
34 0034 1  
35 0035 1 ABSTRACT:  
36 0036 1  
37 0037 1 This routine creates a new file ID by allocating a file number from the  
38 0038 1 index file bitmap. It returns an empty file header, verified for use.  
39 0039 1  
40 0040 1 ENVIRONMENT:  
41 0041 1  
42 0042 1 STARLET operating system, including privileged system services  
43 0043 1 and internal exec routines.  
44 0044 1  
45 0045 1 --  
46 0046 1  
47 0047 1  
48 0048 1 AUTHOR: Andrew C. Goldstein, CREATION DATE: 28-Mar-1977 13:49  
49 0049 1  
50 0050 1 MODIFIED BY:  
51 0051 1  
52 0052 1 A0101 ACG0117 Andrew C. Goldstein, 16-Jan-1980 17:07  
53 0053 1 Return true I/O status on I/O errors  
54 0054 1  
55 0055 1 A0100 ACG00001 Andrew C. Goldstein, 10-Oct-1978 20:02  
56 0056 1 Previous revision history moved to F11A.REV  
57 0057 1 **
```

CREHDR
V04-000

K 14
16-Sep-1984 00:54:43
14-Sep-1984 12:29:25

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[F11A.SRC]CREHDR.B32;1 Page (1) 2

```

: 58      0058 1
: 59      0059 1
: 60      0060 1 LIBRARY 'SYSS$LIBRARY:LIB.L32';
: 61      0061 1 REQUIRE 'SRC$:FCPDEF.B32';
: 62      0376 1
: 63      0377 1
: 64      0378 1 FORWARD ROUTINE
: 65      0379 1   CREATE_HEADER,
: 66      0380 1   UPDATE_IBVBN   : NOVALUE,
: 67      0381 1   READ_NEW_HEADER,
: 68      0382 1   HANDLER;
```

```
! create file ID and header
! update index bitmap scan start
! read new file header block
! local condition handler
```



```

70 0383 1 GLOBAL ROUTINE CREATE_HEADER =
71 0384 1
72 0385 1 ++
73 0386 1
74 0387 1 FUNCTIONAL DESCRIPTION:
75 0388 1
76 0389 1 This routine creates a new file ID by searching the volume's index
77 0390 1 file bitmap for the first free file number. It also checks that a header
78 0391 1 for the file number is present in the index file. It reads the old
79 0392 1 header and establishes the file sequence number for the new one.
80 0393 1
81 0394 1 CALLING SEQUENCE:
82 0395 1 CREATE_HEADER ()
83 0396 1
84 0397 1 INPUT PARAMETERS:
85 0398 1 NONE
86 0399 1
87 0400 1 IMPLICIT INPUTS:
88 0401 1 CURRENT_VCB: address of volume's VCB
89 0402 1
90 0403 1 OUTPUT PARAMETERS:
91 0404 1 NONE
92 0405 1
93 0406 1 IMPLICIT OUTPUTS:
94 0407 1 NONE
95 0408 1
96 0409 1 ROUTINE VALUE:
97 0410 1 address of buffer containing new header
98 0411 1
99 0412 1 SIDE EFFECTS:
100 0413 1 VCB and index file bitmap altered, header block read
101 0414 1
102 0415 1 --
103 0416 1
104 0417 2 BEGIN
105 0418 2
106 0419 2 LOCAL
107 0420 2 VCB : REF BBLOCK, : local copy of VCB address
108 0421 2 VBN, : relative block number in bitmap
109 0422 2 BUFFER : REF BITVECTOR, : address of index file bitmap buffer
110 0423 2 ADDRESS : REF BITVECTOR, : address of byte in buffer
111 0424 2 BITPOS, : bit position of free bit within byte
112 0425 2 FILE_NUMBER, : file number allocated
113 0426 2 IDX_FCB : REF BBLOCK, : FCB of index file
114 0427 2 LBN, : LBN of new file header
115 0428 2 HEADER : REF BBLOCK, : address of header buffer
116 0429 2 SAVE_STATUS; : save I/O status during CHECK_HEADER call
117 0430 2
118 0431 2 EXTERNAL
119 0432 2 CURRENT_VCB : REF BBLOCK, : VCB of volume
120 0433 2 NEW_FID, : pending file ID
121 0434 2 HEADER_LBN, : LBN of created file header
122 0435 2 USER_STATUS : VECTOR; : I/O status block of user
123 0436 2
124 0437 2 EXTERNAL ROUTINE
125 0438 2 READ_BLOCK, : read block from disk
126 0439 2 WRITE_BLOCK, : write block to disk
```

```

127 0440 2      EXTEND INDEX,      ! extend the index file
128 0441 2      MAP_VBN,        ! map virtual to logical block
129 0442 2      CHECK_HEADER;   ! verify file header
130 0443 2
131 0444 2
132 0445 2 ! The outer loop performs retries if blocks in the index file are bad.
133 0446 2 ! Bad header blocks are simply left marked in use in the index file bitmap;
134 0447 2 ! they will show up in a verify but are otherwise harmless.
135 0448 2
136 0449 2
137 0450 2 VCB = .CURRENT_VCB;
138 0451 2 WHILE 1 DO
139 0452 2 BEGIN
140 0453 2
141 0454 2 ! We scan the index file bitmap for the first free (zero) bit. This is done
142 0455 2 ! by starting with the block recorded in the VCB and looking at each block
143 0456 2 ! with a character scan.
144 0457 2
145 0458 2
146 0459 2 VBN = .VCB[VCB$B_IBMAPVBN];
147 0460 2
148 0461 2 IF
149 0462 2 BEGIN
150 0463 2 UNTIL .VBN GEQ .VCB[VCB$B_IBMAPSIZE] DO
151 0464 2 BEGIN
152 0465 2 BUFFER = READ_BLOCK (.VBN + .VCB[VCB$L_IBMAPLBN], 1, INDEX TYPE);
153 0466 2 IF NOT CH$FAIC (ADDRESS = CH$FIND_NOT_CH (512, .BUFFER, 255))
154 0467 2 THEN EXITLOOP 0;
155 0468 2 VBN = .VBN + 1;
156 0469 2 END
157 0470 2 END
158 0471 2
159 0472 2 ! If we fall through the loop, the entire bitmap is full.
160 0473 2
161 0474 2
162 0475 2 THEN ERR_EXIT (SS$_IDXFILEFULL);
163 0476 2
164 0477 2 ! Having found a byte containing a zero bit, scan for the bit.
165 0478 2
166 0479 2
167 0480 2 FFC (%REF (0), %REF (8), .ADDRESS, BITPOS);
168 0481 2
169 0482 2 ! Compute the file number and check it against the maximum files allowed
170 0483 2 ! on the volume. Also check if the corresponding file header is present in
171 0484 2 ! the index file. If not, extend the index file and re-read the bitmap
172 0485 2 ! block, which may have been kicked out in the process.
173 0486 2
174 0487 2
175 0488 2 FILE_NUMBER = .VBN*4096 + (.ADDRESS-.BUFFER)*8 + .BITPOS + 1;
176 0489 2
177 0490 2 IF .FILE_NUMBER GTR .VCB[VCB$L_MAXFILES]
178 0491 2 THEN ERR_EXIT (SS$_IDXFILEFULL);
179 0492 2
180 0493 2 IDX_FCB = .VCB[VCB$L_FCBFL];
181 0494 2 IF .FILE_NUMBER + .VCB[VCB$B_IBMAPSIZE] + 2 GTR .IDX_FCB[FCB$L_FILESIZE]
182 0495 2 THEN
183 0496 2 BEGIN
```



```
184 0497 4 ADDRESS = .ADDRESS - .BUFFER;
185 0498 4 EXTEND_INDEX (.FILE_NUMBER);
186 0499 4 BUFFER = READ_BLOCK (.VBN + .VCB[VCB$L_IBMAPLBN], 1, INDEX_TYPE);
187 0500 4 ADDRESS = .ADDRESS + .BUFFER;
188 0501 4 END;
189 0502 4
190 0503 4 ! All is in order. Set the bit and rewrite the block. Also update the
191 0504 4 scan point in the VCB for the next create. Note that if the file number was
192 0505 4 from the last bit in the block, we will start the scan at the next block to
193 0506 4 avoid wasting a read.
194 0507 4
195 0508 4
196 0509 4 ADDRESS[.BITPOS] = 1;
197 0510 4 WRITE_BLOCK (.BUFFER);
198 0511 4
199 0512 4 IF .FILE_NUMBER<0,12> EQL 0
200 0513 4 THEN VBN = .VBN + 1;
201 0514 4 KERNEL_CALL (UPDATE_IBVBN, .VBN);
202 0515 4 NEW_FID = .FILE_NUMBER; ! record file ID for cleanup
203 0516 4
204 0517 4 ! Now read the old file header. If the block contained an old file header,
205 0518 4 bump the file sequence number; else assign 1.
206 0519 4
207 0520 4
208 0521 4 VBN = .FILE_NUMBER + .VCB[VCB$B_IBMAPSIZE] + 2;
209 0522 4 IDX_FCB = .VCB[VCB$L_FCBFL];
210 0523 4 LBN = MAP_VBN (.VBN, .IDX_FCB[FCB$L_WFL]);
211 0524 4 IF .LBN EQL -1 THEN BUG_CHECK (HDRNOTMAP, FATAL, 'Allocated file header not mapped');
212 0525 4 HEADER = READ_NEW_HEADER (.LBN);
213 0526 4 IF .HEADER NEQ 0 THEN EXITLOOP;
214 0527 4 END; ! end of file number allocation loop
215 0528 4
216 0529 4 HEADER_LBN = .LBN; ! record LBN of new header
217 0530 4
218 0531 4 SAVE_STATUS = .USER_STATUS[0];
219 0532 4 IF CHECK_HEADER (.HEADER, UPLIT_WORD (0, 0, 0)) NEQ 0
220 0533 4 THEN HEADER[FH1$W_FID_SEQ] = .HEADER[FH1$W_FID_SEQ] + 1
221 0534 4 ELSE HEADER[FH1$W_FID_SEQ] = 1;
222 0535 4 HEADER[FH1$W_FID_NUM] = .FILE_NUMBER;
223 0536 4 USER_STATUS[0] = .SAVE_STATUS; ! restore status, bashed by CHECK_HEADER
224 0537 4
225 0538 4 RETURN .HEADER;
226 0539 4
227 0540 1 END; ! end of routine CREATE_HEADER
```

```
.TITLE CREHDR
.IDENT \V04-000\
.PSECT $CODE$,NOWRT,2
```

```
0000 0000 0000 00000 P.AAA: .WORD 0, 0, 0
```

```
.EXTRN CURRENT_VCB, NEW_FID
.EXTRN HEADER_LBN, USER_STATUS
.EXTRN READ_BLOCK, WRITE_BLOCK
.EXTRN EXTEND_INDEX, MAP_VBN
```


.ENTRY	CREATE HEADER, Save R2,R3,R4,R5,R6,R7,R8,-	0383
	R9,R10,R11	
MOVL	CURRENT_VCB, VCB	0450
MOVZBL	58(VCB), VBN	0459
CMPZV	#0, #8, 56(VCB), VBN	0463
BLEQ	5\$	
PUSHL	#3	0465
PUSHL	#1	
PUSHAB	@48(VCB)[VBN]	
CALLS	#3, READ_BLOCK	
MOVL	R0, BUFFER	
SKPC	#255, #512, (BUFFER)	0466
BNEQ	3\$	
CLRL	R1	
MOVL	R1, ADDRESS	
BNEQ	4\$	
INCL	VBN	0468
BRB	2\$	0463
FFC	#0, #8, (ADDRESS), BITPOS	0480
ASHL	#12, VBN, R0	0488
SUBL3	BUFFER, ADDRESS, R1	
MOVAQ	(R0)[R1], R0	
MOVAB	1(BITPOS)[R0], FILE_NUMBER	
CML	FILE_NUMBER, 68(VCB)	0490
BLEQ	6\$	
CHMU	#2256	0491
RET		
MOVL	(VCB), IDX_FCB	0493
MOVZBL	56(VCB), R0	0494
MOVAB	2(R0)[FILE_NUMBER], R7	
CML	R7, 56(IDX_FCB)	
BLEQ	7\$	
SUBL2	BUFFER, ADDRESS	0497
PUSHL	FILE_NUMBER	0498
CALLS	#1, EXTEND_INDEX	
PUSHL	#3	0499
PUSHL	#1	
PUSHAB	@48(VCB)[VBN]	
CALLS	#3, READ_BLOCK	
MOVL	R0, BUFFER	
ADDL2	BUFFER, ADDRESS	0500
BBSS	BITPOS, (ADDRESS), 8\$	0509
PUSHL	BUFFER	0510
CALLS	#1, WRITE_BLOCK	
BITW	FILE_NUMBER, #4095	0512
BNEQ	9\$	
INCL	VBN	0513
PUSHL	VBN	0514
PUSHL	#1	
PUSHL	SP	
PUSHAB	UPDATE IBVBN	
CALLS	#4, @#SYSS\$CMKRN	
MOVL	FILE_NUMBER, NEW_FID	0515
MOVL	R7, VBN	0521

CREHDR
V04-000

C 15
16-Sep-1984 00:54:43
14-Sep-1984 12:29:25

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[F11A.SRC]CREHDR.B32;1

Page 7
(2)

	56		63	D0	000B4	MOVL	(VCB), IDX_FCB	:	0522	
		10	A6	DD	000B7	PUSHL	16(IDX_FCB)	:	0523	
			52	DD	000BA	PUSHL	VCB	:		
0000G	CF		02	FB	000BC	CALLS	#2, MAP_VBN	:		
FFFFFFF	5B		50	D0	000C1	MOVL	R0, LBN	:		
	8F		5B	D1	000C4	CMPL	LBN, #-1	:	0524	
			04	12	000CB	BNEQ	10\$	:		
				FEFF	000CD	BUGW		:		
				0000*	000CF	.WORD	<BUG\$_HDRNOTMAP!4>	:		
			5B	DD	000D1	10\$:	PUSHL	LBN	:	0525
0000V	CF		01	FB	000D3	CALLS	#1, READ_NEW_HEADER	:		
	54		50	D0	000D8	MOVL	R0, HEADER	:		
			03	12	000DB	BNEQ	11\$	:	0526	
			FF27	31	000DD	BRW	1\$	:		
0000G	CF		5B	D0	000E0	11\$:	MOVL	LBN, HEADER_LBN	:	0529
	52	0000G	CF	D0	000E5	MOVW	USER_STATUS, SAVE_STATUS	:	0531	
		FF0C	CF	9F	000EA	PUSHAB	P.AAX	:	0532	
			54	DD	000EE	PUSHL	HEADER	:		
0000G	CF		02	FB	000F0	CALLS	#2, CHECK_HEADER	:		
			50	D5	000F5	TSTL	R0	:		
			05	13	000F7	BEQL	12\$	:		
		04	A4	B6	000F9	INCW	4(HEADER)	:	0533	
			04	11	000FC	BRB	13\$	:		
04	A4		01	B0	000FE	12\$:	MOVW	#1, 4(HEADER)	:	0534
02	A4		55	B0	00102	13\$:	MOVW	FILE_NUMBER, 2(HEADER)	:	0535
0000G	CF		52	D0	00106	MOVL	SAVE_STATUS, USER_STATUS	:	0536	
	50		54	D0	0010B	MOVL	HEADER, R0	:	0538	
			04	0010E	RET			:	0540	

; Routine Size: 271 bytes, Routine Base: \$CODE\$ + 0006


```

: 229 0541 1 GLOBAL ROUTINE UPDATE_IBVBN (VBN) : NOVALUE =
: 230 0542 1
: 231 0543 1 ++
: 232 0544 1
: 233 0545 1 FUNCTIONAL DESCRIPTION:
: 234 0546 1
: 235 0547 1 This routine writes back the starting VBN for the index file bitmap
: 236 0548 1 scan into the vcb. This routine must be called in kernel mode.
: 237 0549 1
: 238 0550 1 CALLING SEQUENCE:
: 239 0551 1 UPDATE_VBN (ARG1)
: 240 0552 1
: 241 0553 1 INPUT PARAMETERS:
: 242 0554 1 ARG1: new start VBN
: 243 0555 1
: 244 0556 1 IMPLICIT INPUTS:
: 245 0557 1 CURRENT_VCB: VCB of volume
: 246 0558 1
: 247 0559 1 OUTPUT PARAMETERS:
: 248 0560 1 NONE
: 249 0561 1
: 250 0562 1 IMPLICIT OUTPUTS:
: 251 0563 1 NONE
: 252 0564 1
: 253 0565 1 ROUTINE VALUE:
: 254 0566 1 NONE
: 255 0567 1
: 256 0568 1 SIDE EFFECTS:
: 257 0569 1 VBN written into VCB
: 258 0570 1
: 259 0571 1 --
: 260 0572 1
: 261 0573 2 BEGIN
: 262 0574 2
: 263 0575 2 EXTERNAL
: 264 0576 2 CURRENT_VCB : REF BBLOCK; ! VCB of volume
: 265 0577 2
: 266 0578 2 CURRENT_VCB[VCB$B_IBMAPVBN] = .VBN;
: 267 0579 2
: 268 0580 1 END; ! end of routine UPDATE_VBN
```

```

          50      0000G CF D0 00002
3A      A0      04  AC  90 00007
          04 0000C
```

```

.ENTRY  UPDATE_IBVBN, Save nothing
MOVL    CURRENT_VCB, R0
MOVB    VBN, 58(R0)
RET
```

```

: 0541
: 0578
:
: 0580
```

; Routine Size: 13 bytes, Routine Base: \$CODE\$ + 0115


```
270 0581 1 ROUTINE READ_NEW_HEADER (LBN) =
271 0582 1
272 0583 1 ++
273 0584 1
274 0585 1 FUNCTIONAL DESCRIPTION:
275 0586 1
276 0587 1 This routine reads the block about to be used for a new file header.
277 0588 1 It uses a local condition handler to fix up errors.
278 0589 1
279 0590 1
280 0591 1 CALLING SEQUENCE:
281 0592 1 READ_NEW_HEADER (ARG1)
282 0593 1
283 0594 1 INPUT PARAMETERS:
284 0595 1 ARG1: LBN of block to read
285 0596 1
286 0597 1 IMPLICIT INPUTS:
287 0598 1 NONE
288 0599 1
289 0600 1 OUTPUT PARAMETERS:
290 0601 1 NONE
291 0602 1
292 0603 1 IMPLICIT OUTPUTS:
293 0604 1 NONE
294 0605 1
295 0606 1 ROUTINE VALUE:
296 0607 1 address of buffer containing block or 0 if bad
297 0608 1
298 0609 1 SIDE EFFECTS:
299 0610 1 block read and/or written
300 0611 1
301 0612 1 --
302 0613 1
303 0614 2 BEGIN
304 0615 2
305 0616 2 LOCAL
306 0617 2 HEADER : REF BBLOCK; ! address of block read
307 0618 2
308 0619 2 EXTERNAL ROUTINE
309 0620 2 READ_BLOCK, ! read a block
310 0621 2 WRITE_BLOCK, ! write a block
311 0622 2 INVALIDATE, ! invalidate a buffer
312 0623 2 CREATE_BLOCK; ! create a new block buffer
313 0624 2
314 0625 2 ! Under control of the condition handler, we read the block. If the read
315 0626 2 ! fails, we attempt to rewrite the block and then read it again. If either
316 0627 2 ! of the latter fails, we return failure.
317 0628 2
318 0629 2
319 0630 2 ENABLE HANDLER;
320 0631 2
321 0632 2 HEADER = READ_BLOCK (.LBN, 1, HEADER_TYPE);
322 0633 2
323 0634 2 IF .HEADER EQL 0
324 0635 2 THEN
325 0636 2 BEGIN
326 0637 2 HEADER = CREATE_BLOCK (.LBN, 1, HEADER_TYPE);
```

```

: 327      0638 3  WRITE_BLOCK (.HEADER);
: 328      0639 3  INVALIDATE (.HEADER);
: 329      0640 3  HEADER = READ_BLOCK (.LBN, 1, HEADER_TYPE);
: 330      0641 2  END;
: 331      0642 2
: 332      0643 2  RETURN .HEADER;
: 333      0644 2
: 334      0645 1  END;
```

! end of routine READ_NEW_HEADER

.EXTRN INVALIDATE, CREATE_BLOCK

```

                                0004 00000 READ_NEW_HEADER:
                                .WORD  Save R2
                                MOVAL  2$, (FP)
                                MOVQ   #1, -(SP)
                                PUSHL  LBN
                                CALLS  #3, READ_BLOCK
                                MOVL   R0, HEADER
                                BNEQ   1$
                                MOVQ   #1, -(SP)
                                PUSHL  LBN
                                CALLS  #3, CREATE_BLOCK
                                MOVL   R0, HEADER
                                PUSHL  HEADER
                                CALLS  #1, WRITE_BLOCK
                                PUSHL  HEADER
                                CALLS  #1, INVALIDATE
                                MOVQ   #1, -(SP)
                                PUSHL  LBN
                                CALLS  #3, READ_BLOCK
                                MOVL   R0, HEADER
                                MOVL   HEADER, R0
                                RET
                                0000 00045 2$:
                                .WORD  Save nothing
                                CLRL   -(SP)
                                PUSHL  SP
                                MOVQ   4(AP), -(SP)
                                CALLS  #3, HANDLER
                                RET
                                04 00054
```

6D	003F	CF	DE	00002				0581
7E		01	7D	00007				0614
	04	AC	DD	0000A				0632
0000G	CF	03	FB	0000D				
52		50	D0	00012				
		2A	12	00015				0634
7E		01	7D	00017				0637
	04	AC	DD	0001A				
0000G	CF	03	FB	0001D				
52		50	D0	00022				
		52	DD	00025				0638
0000G	CF	01	FB	00027				
		52	DD	0002C				0639
0000G	CF	01	FB	0002E				
7E		01	7D	00033				0640
	04	AC	DD	00036				
0000G	CF	03	FB	00039				
52		50	D0	0003E				
50		52	D0	00041 1\$:				0643
		04	00044					0645
		0000	00045 2\$:					0614
		7E	D4	00047				
		5E	DD	00049				
7E	04	AC	7D	0004B				
0000V	CF	03	FB	0004F				
		04	00054					

; Routine Size: 85 bytes, Routine Base: \$CODE\$ + 0122


```
0646 1 ROUTINE HANDLER (SIGNAL, MECHANISM) =
0647 1
0648 1 ++
0649 1
0650 1 FUNCTIONAL DESCRIPTION:
0651 1
0652 1 This routine is the condition handler for the initial header read.
0653 1 On surface errors, it unwinds and causes a return of 0 to the caller
0654 1 of the I/O routine to indicate error. Hard drive errors cause the
0655 1 usual error exit.
0656 1
0657 1 CALLING SEQUENCE:
0658 1 HANDLER (ARG1, ARG2)
0659 1
0660 1 INPUT PARAMETERS:
0661 1 ARG1: address of signal array
0662 1 ARG2: address of mechanism array
0663 1
0664 1 IMPLICIT INPUTS:
0665 1 NONE
0666 1
0667 1 OUTPUT PARAMETERS:
0668 1 NONE
0669 1
0670 1 IMPLICIT OUTPUTS:
0671 1 NONE
0672 1
0673 1 ROUTINE VALUE:
0674 1 $$$_RESIGNAL or none if unwind
0675 1
0676 1 SIDE EFFECTS:
0677 1 NONE
0678 1
0679 1 --
0680 1
0681 1
0682 2 BEGIN
0683 2
0684 2 MAP
0685 2 SIGNAL : REF BBLOCK, ! signal arg array
0686 2 MECHANISM : REF BBLOCK; ! mechanism arg array
0687 2
0688 2 EXTERNAL
0689 2 IO_STATUS : VECTOR; ! I/O status block of last operation
0690 2
0691 2
0692 2 ! If the condition is change mode to user (error exit) and the status is
0693 2 ! read error, zero the return R0 and unwind to the the establisher. On
0694 2 ! most write errors, zero the return R0 and unwind to the caller.
0695 2 ! Otherwise, just resignal the condition.
0696 2
0697 2
0698 2 IF .SIGNAL[CHFS$_SIG_NAME] EQL $$$_CMODUSER
0699 2 THEN
0700 3 BEGIN
0701 3 MECHANISM[CHFS$_MCH_SAVR0] = 0;
0702 3
```

CREHDR
V04-000

H 15
16-Sep-1984 00:54:43
14-Sep-1984 12:29:25

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[F11A.SRC]CREHDR.B32;1 Page 12
(5)

```

: 393      0703 3      IF .SIGNAL[CHFSL_SIG_ARG1] EQL SSS_PARITY
: 394      0704 3      OR .SIGNAL[CHFSL_SIG_ARG1] EQL SSS_DATACHECK
: 395      0705 3      OR .SIGNAL[CHFSL_SIG_ARG1] EQL SSS_FORMAT
: 396      0706 3      THEN
: 397      0707 4      $UNWIND (DEPADR = MECHANISM[CHFSL_MCH_DEPTH])
: 398      0708 2      END;
: 399      0709 2
: 400      0710 2      RETURN SSS_RESIGNAL;      ! status is irrelevant if unwinding
: 401      0711 2
: 402      0712 1      END;      ! end of routine HANDLER

```

```

                                .EXTRN IO_STATUS, SYSSUNWIND
                                0000 00000 HANDLER:.WORD Save nothing
                                00000424 8F 04 AC D0 00002 MOVL SIGNAL, R0      : 0646
                                00000424 8F 04 AO D1 00006 CMPL 4(R0), #1060    : 0698
                                00000424 8F 04 31 12 0000E BNEQ 2$
                                00000424 8F 08 AC D0 00010 MOVL MECHANISM, R1    : 0701
                                00000424 8F 08 0C A1 D4 00014 CLRL 12(R1)
                                000001F4 8F 08 AO D1 00017 CMPL 8(R0), #500    : 0703
                                000001F4 8F 08 14 13 0001F BEQL 1$
                                0000005C 8F 08 AO D1 00021 CMPL 8(R0), #92     : 0704
                                0000005C 8F 08 0A 13 00029 BEQL 1$
                                000000BC 8F 08 AO D1 0002B CMPL 8(R0), #188    : 0705
                                000000BC 8F 08 0C 12 00033 BNEQ 2$
                                000000BC 8F 08 7E D4 00035 1$: CLRL -(SP)      : 0707
                                00000000G 00 08 A1 9F 00037 PUSHAB 8(R1)
                                00000000G 00 02 FB 0003A CALLS #2, SYSSUNWIND
                                00000000G 50 0918 8F 3C 00041 2$: MOVZWL #2328, R0
                                00000000G 50 04 00046 RET                      : 0710
                                : 0712

```

; Routine Size: 71 bytes, Routine Base: \$CODE\$ + 0177

```

: 403      0713 1
: 404      0714 1 END
: 405      0715 0 ELUDOM

```

PSECT SUMMARY

Name	Bytes	Attributes
\$CODE\$	446	NOVEC,NOWRT, RD , EXE,NOSHR, LCL, REL, CON,NOPIC,ALIGN(2)

Library Statistics

----- Symbols -----	Pages	Processing
---------------------	-------	------------

CREHDR
V04-000

I 15
16-Sep-1984 00:54:43
14-Sep-1984 12:29:25

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[F11A.SRC]CREHDR.B32;1 Page 13
(5)

File	Total	Loaded	Percent	Mapped	Time
_\$255\$DUA28:[SYSLIB]LIB.L32;1	18619	22	0	1000	00:01.8

COMMAND QUALIFIERS

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

; Size: 440 code + 6 data bytes
; Run Time: 00:12.0
; Elapsed Time: 00:36.4
; Lines/CPU Min: 3586
; Lexemes/CPU-Min: 13219
; Memory Used: 139 pages
; Compilation Complete

0164

**DIGITAL
CONFIDE**

EQUIPMENT
INITIAL AND

CORPORATION
PROPRIETARY