


```

CCCCCCCC RRRRRRRR EEEEEEEEE H H DDDDDDD RRRRRRR
CCCCCCCC RRRRRRRR EEEEEEEEE H H DDDDDDD RRRRRRR
CC        RR      RR      EE      HH      HH      DD      DC RR      RR
CC        RR      RR      EE      HH      HH      DD      DD RR      RR
CC        RR      RR      EE      HH      HH      DD      DD RR      RR
CC        RRRRRRRR EEEEEEEE HHHHHHHHH DD      DD RRRRRRR
CC        RRRRRRRR EEEEEEEE HHHHHHHHH DD      DD RRRRRRR
CC        RR  RR   EE      HH      HH      DD      DD RR  RR
CC        RR  RR   EE      HH      HH      DD      DD RR  RR
CC        RR      RR  EE      HH      HH      DD      DD RR      RR
CC        RR      RR  EE      HH      HH      DD      DD RR      RR
CCCCCCCC RR      RR  EEEEEEEEE HH      HH      DDDDDDD RR      RR
CCCCCCCC RR      RR  EEEEEEEEE HH      HH      DDDDDDD RR      RR

```

```

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
LL        II      SS
LLLLLLLLLL IIIIII SSSSSSSS
LLLLLLLLLL IIIIII SSSSSSSS

```

1
2
:001 :CDS0018
4-1
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
:001 :CDS0018
:002 :CDS0018
:003 :CDS0018
52
53
54

```

0001 0 MODULE CREHDR (
0002 0     LANGUAGE (BLISS32),
0003 0     IDENT = 'V04-001'
0004 0 ) =
0005 1 BEGIN
0006 1
0007 1
0008 1 *****
0009 1 *
0010 1 *  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0011 1 *  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0012 1 *  ALL RIGHTS RESERVED.
0013 1 *
0014 1 *  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0015 1 *  ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0016 1 *  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0017 1 *  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0018 1 *  OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0019 1 *  TRANSFERRED.
0020 1 *
0021 1 *  THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0022 1 *  AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0023 1 *  CORPORATION.
0024 1 *
0025 1 *  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0026 1 *  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0027 1 *
0028 1 *
0029 1 *****
0030 1
0031 1 ++
0032 1
0033 1 FACILITY: F11ACP Structure Level 2
0034 1
0035 1 ABSTRACT:
0036 1
0037 1     This routine creates a new file ID by allocating a file number from the
0038 1     index file bitmap. It returns an empty file header, verified for use.
0039 1
0040 1 ENVIRONMENT:
0041 1
0042 1     STARLET operating system, including privileged system services
0043 1     and internal exec routines.
0044 1
0045 1 --
0046 1
0047 1
0048 1 AUTHOR: Andrew C. Goldstein, CREATION DATE: 28-Mar-1977 13:49
0049 1
0050 1 MODIFIED BY:
0051 1
0052 1     V04-001 CDS0018      Christian D. Saether    15-Nov-1984
0053 1     Expand test for clusterness to look at clu$gl_club.
0054 1
0055 1     V03-022 CDS0017      Christian D. Saether    20-Aug-1984
0056 1     Force fcb for indexf to be stale always before
0057 1     attempting to map vbns.

```

55	0058	1	
56	0059	1	V03-021 CDS0016 Christian D. Saether 13-Aug-1984
57	0060	1	Back off an extra dot in ACG0438.
58	0061	1	
59	0062	1	V03-020 ACG0438 Andrew C. Goldstein, 1-Aug-1984 11:55
60	0063	1	Add cache interlock logic on FID cache; use central
61	0064	1	dequeue routine.
62	0065	1	
63	0066	1	V03-019 LMP0278 L. Mark Pilant, 12-Jul-1984 10:58
64	0067	1	Fix a bug that caused the EXBYFLM error if it was necessary
65	0068	1	to turn the index file window.
66	0069	1	
67	0070	1	V03-018 CDS0015 Christian D. Saether 17-Apr-1984
68	0071	1	Have MAP_IDX check to see whether curr_lckindx is
69	0072	1	for the index file to avoid releasing it if so.
70	0073	1	
71	0074	1	V03-017 CDS0014 Christian D. Saether 11-Apr-1984
72	0075	1	Release allocation lock prior to serializing on
73	0076	1	new primary header. This eliminates potential
74	0077	1	deadlocks when the new primary header is a valid
75	0078	1	header that someone else is messing with.
76	0079	1	
77	0080	1	V03-016 CDS0013 Christian D. Saether 1-Apr-1984
78	0081	1	ACG0409 forgot to rewrite indexf bitmap buffer. No joke.
79	0082	1	
80	0083	1	V03-015 ACG0409 Andrew C. Goldstein, 21-Mar-1984 19:40
81	0084	1	Redesign file ID cacheing algorithm so that file ID's
82	0085	1	beyond the index file EOF are not cached. Eliminate
83	0086	1	BASH_HEADERS routine; general code cleanup to remove
84	0087	1	kernel calls. CHECK_HEADER2 no longer writes USER_STATUS.
85	0088	1	
86	0089	1	V03-014 ACG0404 Andrew C. Goldstein, 15-Mar-1984 17:37
87	0090	1	Correct releasing of file sync lock when retrying for a header
88	0091	1	
89	0092	1	V03-013 CDS0012 Christian D. Saether 23-Feb-1984
90	0093	1	Eliminate references to FLUSH_LOCK_BASIS.
91	0094	1	
92	0095	1	V03-012 CDS0011 Christian D. Saether 27-Dec-1983
93	0096	1	Use BIND_COMMON macro.
94	0097	1	
95	0098	1	V03-011 CDS0010 Christian D. Saether 12-Dec-1983
96	0099	1	Start of XQP code is at symbol INITXQP now.
97	0100	1	
98	0101	1	V03-010 CDS0009 Christian D. Saether 5-Oct-1983
99	0102	1	Fix bug restoring privileges to the PCB.
100	0103	1	
101	0104	1	V03-009 CDS0008 Christian D. Saether 3-Oct-1983
102	0105	1	Save/restore CURR_LCKINDX where necessary rather
103	0106	1	than PRIM_LCKINDX.
104	0107	1	
105	0108	1	V03-008 CDS0007 Christian D. Saether 13-Sep-1983
106	0109	1	Modify interface to allocation serialization.
107	0110	1	
108	0111	1	V03-007 CDS0006 Christian D. Saether 12-May-1983
109	0112	1	Serialize header creation.
110	0113	1	
111	0114	1	V03-006 CDS0005 Christian D. Saether 1-Mar-1983

```

: 112      0115 1  :
: 113      0116 1  :
: 114      0117 1  :
: 115      0118 1  :
: 116      0119 1  :
: 117      0120 1  :
: 118      0121 1  :
: 119      0122 1  :
: 120      0123 1  :
: 121      0124 1  :
: 122      0125 1  :
: 123      0126 1  :
: 124      0127 1  :
: 125      0128 1  :
: 126      0129 1  :
: 127      0130 1  :
: 128      0131 1  :
: 129      0132 1  :
: 130      0133 1  :
: 131      0134 1  :
: 132      0135 1  :
: 133      0136 1  :
: 134      0137 1  :
: 135      0138 1  :
: 136      0139 1  :
: 137      0140 1  :
: 138      0141 1  :
: 139      0142 1  :
: 140      0143 1  :
: 141      0144 1  :
: 142      0145 1  :
: 143      0146 1  :
: 144      0147 1  :
: 145      0148 1  :
: 146      1139 1  :
: 147      1140 1  :
: 148      1141 1  :
: 149      1142 1  :
: 150      1143 1  :
: 151      1144 1  :
: 152      1145 1  :
: 153      1146 1  :
: 154      1147 1  :
: 155      1148 1  :

      Need BYPASS privilege also.

V03-005 CDS0004      Christian D. Saether      20-Feb-1983
      Call MAP_VBN before checking FILESIZE so that
      header is checked before deciding to extend
      index file.
      Also make READ_IDX_HEADER insensitive to headers that
      map more than the FCB knows about.
      Totally punt figuring out what to do with EFBLK
      for the index file.

V03-004 CDS0003      Christian D. Saether      13-Jan-1983
      Separately save and restore PHD privs.

V03-003 CDS0002      Christian D. Saether      28-Dec-1982
      Give priv around QIO.

V03-002 CDS0001      C Saether                  3-Aug-1982
      Change QIOW to QIO with completion AST.

V03-001 ACG0273      Andrew C. Goldstein,      23-Mar-1982 10:50
      Use random file sequence number if old header is junk,
      use alternate index file header if primary is suspect

V02-007 ACG0229      Andrew C. Goldstein,      23-Dec-1981 21:53
      Count file ID cache hits and misses

V02-006 ACG0167      Andrew C. Goldstein,      16-Apr-1980 19:25
      Previous revision history moved to F11B.REV

      **

LIBRARY 'SYSSLIBRARY:LIB.L32';
REQUIRE 'SRCS:FCPDEF.B32';

FORWARD ROUTINE
CREATE_HEADER : L_NORM, ! create file ID and header
FILL_FID_CACHE : L_NORM NOVALUE, ! load file ID cache from bitmap
INIT_FID_CACHE : L_NORM NOVALUE, ! initialize file ID cache lock
READ_NEW_HEADER : L_NORM, ! read new file header block
HANDLER, ! local condition handler
READ_IDX_HEADER : L_NORM, ! read index file header
MAP_IDX : L_NORM; ! map vbn for index file.

```

```

157 1149 1 GLOBAL ROUTINE CREATE_HEADER (FILE_ID) : L_NORM =
158 1150 1
159 1151 1 ++
160 1152 1
161 1153 1 FUNCTIONAL DESCRIPTION:
162 1154 1
163 1155 1     This routine creates a new file ID by searching the volume's index
164 1156 1     file bitmap for the first free file number. It also checks that a
165 1157 1     header for the file number is present in the index file. It reads
166 1158 1     the old header and establishes the file sequence number for the
167 1159 1     new one.
168 1160 1
169 1161 1 CALLING SEQUENCE:
170 1162 1     CREATE_HEADER (ARG1)
171 1163 1
172 1164 1 INPUT PARAMETERS:
173 1165 1     NONE
174 1166 1
175 1167 1 IMPLICIT INPUTS:
176 1168 1     CURRENT_VCB: address of volume's VCB
177 1169 1
178 1170 1 OUTPUT PARAMETERS:
179 1171 1     ARG1: address to store file ID of created header
180 1172 1
181 1173 1 IMPLICIT OUTPUTS:
182 1174 1     NEW_FID: file number of header created
183 1175 1     NEW_FID_RVN: RVN of above
184 1176 1
185 1177 1 ROUTINE VALUE:
186 1178 1     address of buffer containing new header
187 1179 1
188 1180 1 SIDE EFFECTS:
189 1181 1     VCB and index file bitmap altered, header block read
190 1182 1
191 1183 1 --
192 1184 1
193 1185 2 BEGIN
194 1186 2
195 1187 2 MAP
196 1188 2     FILE_ID          : REF BBLOCK;    ! new file ID of header
197 1189 2
198 1190 2 LABEL
199 1191 2     GET_FILE_NUM;    :                ! acquire a file number
200 1192 2
201 1193 2 LOCAL
202 1194 2     CACHE_FLUSHED,  :                ! flag indicating cluster caches flushed
203 1195 2     NEW_LCKINDX     : INITIAL (0),
204 1196 2     TEMP,           :                ! temp storage for current lock index
205 1197 2     VCB              : REF BBLOCK,    ! local copy of VCB address
206 1198 2     FID_CACHE       : REF BBLOCK,    ! pointer to file ID cache
207 1199 2     VBN             :                ! relative block number in bitmap
208 1200 2     BUFFER          : REF BITVECTOR, ! address of index file bitmap buffer
209 1201 2     ADDRESS         : REF BITVECTOR, ! address of byte in buffer
210 1202 2     CURRENT_EOF,    :                ! current EOF of index file
211 1203 2     COUNT           :                ! number of index blocks to bash
212 1204 2     FILE_NUMBER,    :                ! file number allocated
213 1205 2     IDX_FCB        : REF BBLOCK,    ! FCB of index file

```

```

: 214      1206      2
: 215      1207      2
: 216      1208      2
: 217      1209      2
: 218      1210      2
: 001 : CDS0018 1211      2
: 219      1212      2
: 220      1213      2
: 221      1214      2
: 222      1215      2
: 223      1216      2
: 224      1217      2
: 225      1218      2
: 226      1219      2
: 227      1220      2
: 228      1221      2
: 229      1222      2
: 230      1223      2
: 231      1224      2
: 232      1225      2
: 233      1226      2
: 234      1227      2
: 235      1228      2
: 236      1229      2
: 237      1230      2
: 238      1231      2
: 239      1232      2
: 240      1233      2
: 241      1234      2
: 242      1235      2
: 243      1236      2
: 244      1237      2
: 245      1238      2
: 246      1239      2
: 247      1240      2
: 248      1241      2
: 249      1242      2
: 250      1243      2
: 251      1244      2
: 252      1245      2
: 253      1246      2
: 254      1247      2
: 255      1248      2
: 256      1249      2
: 257      1250      2
: 258      1251      2
: 259      1252      2
: 260      1253      2
: 261      1254      2
: 262      1255      2
: 263      1256      2
: 264      1257      2
: 265      1258      2
: 266      1259      2
: 267      1260      2
: 268      1261      2
: 269      1262      2

```

LBN,
HEADER : REF BBLOCK, ! LBN of new file header
STATUS: ! address of header buffer
! value of CHECK_HEADER call
EXTERNAL
CLUSGL_CLUB : ADDRESSING_MODE (GENERAL),
PMS\$GL_FIDMIT : ADDRESSING_MODE (GENERAL),
PMS\$GL_FIDMISS : ADDRESSING_MODE (GENERAL),
EXES\$Q_SYSTIME : ADDRESSING_MODE (GENERAL);
! count of file ID cache hits
! count of file ID cache misses
! system time of day
BIND_COMMON;
EXTERNAL ROUTINE
ALLOCATION_LOCK : L_NORM NOVALUE, ! interlock allocation
ALLOCATION_UNLOCK : L_NORM NOVALUE, ! release allocation lock.
SERIAL_FILE : L_NORM, ! serialize file processing
RELEASE_SERIAL_LOCK : L_NORM NOVALUE, ! release processing lock
DEQ_LOCK : L_NORM, ! dequeue a lock
READ_BLOCK : L_NORM, ! read block from disk
WRITE_BLOCK : L_NORM, ! write block to disk
DELETE_FID : L_NORM, ! flush file ID cache and release lock
RELEASE_LOCKBASIS : L_NORM, ! release buffers under specified lock
CACHE_LOCK : L_NORM, ! acquire cache sync lock
EXTEND_INDEX : L_NORM, ! extend the index file
ERASE_BLOCKS : L_NORM, ! erase blocks on disk
CHECKSUM : L_NORM, ! compute file header checksum
WRITE_HEADER : L_NORM, ! write current file header
RESET_LBN : L_NORM, ! change backing LBN of buffer
INVALIDATE : L_NORM, ! invalidate a buffer
CREATE_BLOCK : L_NORM, ! materialize a block buffer
CHECK_HEADER2 : L_NORM, ! verify file header
MARK_DIRTY : L_NORM; ! mark buffer for write-back
! Serialize further file header creation processing.
ALLOCATION_LOCK ();
! The outer loop performs retries if blocks in the index file are bad or
! are valid file headers. A block containing a valid file header is never
! used to create a new file; it is simply left marked in use for recovery.
! Bad header blocks are simply left marked in use in the index file bitmap;
! they will show up in a verify but are otherwise harmless.
VCB = .CURRENT VCB;
FID_CACHE = .BBLOCK [.VCB[VCBSL_CACHE], VCASL_FIDCACHE];
CACHE_FLUSHED = 0;
WHILE -1 DO
GET_FILE_NUM: BEGIN
! See if a file number is available in the file number cache. If not,
! we scan the index file bitmap for the first free (zero) bit. This is done
! by starting with the block recorded in the VCB and looking at each block
! with a character scan.

```
270 1263 3 :
271 1264 3 :
272 1265 3 :
273 1266 3 :
274 1267 4 :
275 1268 4 :
276 1269 4 :
277 1270 4 :
278 1271 4 :
279 1272 5 :
280 1273 5 :
281 1274 6 :
282 1275 6 :
283 1276 6 :
284 1277 6 :
285 1278 6 :
286 1279 6 :
287 1280 5 :
288 1281 5 :
289 1282 5 :
290 1283 5 :
291 1284 5 :
292 1285 5 :
293 1286 5 :
294 1287 4 :
295 1288 4 :
296 1289 4 :
297 1290 5 :
298 1291 5 :
299 1292 5 :
300 1293 5 :
301 1294 5 :
302 1295 5 :
303 1296 5 :
304 1297 5 :
305 1298 5 :
306 1299 5 :
307 1300 5 :
308 1301 5 :
309 1302 6 :
:001 !CDS0018 1303 6 :
:002 !CDS0018 1304 6 :
311-1 1305 6 :
312 1306 6 :
313 1307 7 :
314 1308 7 :
315 1309 7 :
316 1310 7 :
317 1311 7 :
318 1312 7 :
319 1313 7 :
320 1314 7 :
321 1315 7 :
322 1316 7 :
323 1317 7 :
324 1318 7 :
325 1319 7 :

IF .FID_CACHE[VCSW_FIDCOUNT] EQL 0
THEN
  BEGIN
    PMSSGL_FIDMISS = .PMSSGL_FIDMISS + 1;
    VBN = .VCB[VCSB_IBMAPVBN];

    IF NOT
      BEGIN
        UNTIL .VBN GEQ .VCB[VCSB_IBMAPSIZE] DO
          BEGIN
            BUFFER = READ_BLOCK (.VBN + .VCB[VCSL_IBMAPLBN], 1, INDEX TYPE);
            IF NOT CH$FAI[ (ADDRESS = CH$FIND_NOT_CH (512, .BUFFER, 255))
            THEN EXITLOOP 0;
            VBN = .VBN + 1;
          END
        END
      END

      ! Having found a bitmap block with free files in it, attempt to fill the
      ! file ID cache. If it refuses to fill, it's because we're at the index
      ! file EOF.

      THEN FILL FID CACHE (.VCB, .BUFFER, .VBN);
      IF .FID_CACHE[VCSW_FIDCOUNT] EQL 0
      THEN
        BEGIN
          ! If the index file EOF coincides with the physical end of file, we have to
          ! extend the index file. Otherwise, we just have to push the EOF. Before
          ! extending the index file, if we are in a cluster, ask for a cluster-wide
          ! flush of the file ID caches.

          IDX_FCB = .VCB[VCSL_FCBFL];
          CURRENT_EOF = .IDX_FCB[FCBSL_EFBLK];
          IF .CURRENT_EOF GEQU .IDX_FCB[FCBSL_FILESIZE]
          THEN
            BEGIN
              IF .BBLOCK [CURRENT_UCB[UCBSL_DEVCHAR2], DEV$V_CLU]
              AND .CLUSGL CLUB NEQ 0
              AND NOT .CACHE_FLUSHED
              THEN
                BEGIN
                  LOCAL IDX_FILE_ID, LOCK_ID;
                  DELETE FID (0);
                  RELEASE_LOCKBASIS (-1);
                  ALLOCATION_UNLOCK ();
                  IDX_FILE_ID = FID$C_INDEXF OR .CURRENT_VCB[VCSW_RVN] ^ 24;
                  LOCK_ID = 0;
                  CACHE_LOCK (.IDX_FILE_ID, LOCK_ID, 1);
                  ALLOCATION_LOCK (?);
                  DEQ_LOCK (.LOCK_ID);
                  CACHE_FLUSHED = -1;
                  LEAVE GET_FILE_NUM;
                END
            END
          END
        END
      END
    END
  END
```



```
.. 326      1320 6      ELSE
.. 327      1321 6      EXTEND_INDEX ();
.. 328      1322 6      END
.. 329      1323 6
.. 330      1324 6      ! Move the EOF and zero the intervening blocks. Note that this version
.. 331      1325 6      ! of the file system always sets the index file EOF to be physical end
.. 332      1326 6      ! of file, because the index file is zeroed on extend. This code is
.. 333      1327 6      ! present for compatibility with past and future file systems that may
.. 334      1328 6      ! not zero the index file on extend. Serialize activity on the index
.. 335      1329 6      ! file header.
.. 336      1330 6
.. 337      1331 6
.. 338      1332 5      ELSE
.. 339      1333 6      BEGIN
.. 340      1334 6      TEMP = .CURR_LCKINDX;
.. 341      1335 6      SERIAL_FILE (IDX_FCB [FCB$W_FID]);
.. 342      1336 6
.. 343      1337 6      LBN = MAP_IDX (.CURRENT_EOF+1, COUNT);
.. 344      1338 6      ERASE_BLOCKS (.LBN, .COUNT, .IO_CHANNEL);
.. 345      1339 6      CURRENT_EOF = .CURRENT_EOF + .COUNT;
.. 346      1340 6
.. 347      1341 6      HEADER = READ_IDX_HEADER ();
.. 348      1342 6      BBLOCK [HEADER[FH2$W_RECATTR], FAT$E_EFBLK] = ROT (.CURRENT_EOF+1, 16);
.. 349      1343 6      BBLOCK [HEADER[FH2$W_RECATTR], FAT$W_FFBYTE] = 0;
.. 350      1344 6      IF .HEADER [FH2$B_IDOFFSET] GEQU ($BYTEOFFSET (FH2$E_HIGHWATER)+4)/2
.. 351      1345 6      THEN HEADER [FH2$E_HIGHWATER] = .CURRENT_EOF + 1;
.. 352      1346 6
.. 353      1347 6      CHECKSUM (.HEADER);
.. 354      1348 6      WRITE_HEADER ();
.. 355      1349 6      IDX_FCB[FCB$E_EFBLK] = .CURRENT_EOF;
.. 356      1350 6      RESET_LBN (.HEADER, .VCB[VCB$E_IHDR2LBN]);
.. 357      1351 6      WRITE_BLOCK (.HEADER);
.. 358      1352 6      INVALIDATE (.HEADER);
.. 359      1353 6
.. 360      1354 6      RELEASE_SERIAL_LOCK (.CURR_LCKINDX);
.. 361      1355 6      CURR_LCKINDX = .TEMP;
.. 362      1356 6      END;
.. 363      1357 5
.. 364      1358 5      ! Go around the loop to try to allocate a file number again.
.. 365      1359 5
.. 366      1360 5
.. 367      1361 5      LEAVE GET_FILE_NUM;
.. 368      1362 5      END
.. 369      1363 4      ELSE
.. 370      1364 4
.. 371      1365 4      ! We successfully filled the file ID cache from the bitmap. Write back
.. 372      1366 4      ! the index file bitmap buffer.
.. 373      1367 4
.. 374      1368 4
.. 375      1369 4      WRITE_BLOCK (.BUFFER);
.. 376      1370 4
.. 377      1371 4      END
.. 378      1372 4
.. 379      1373 4      ! If the file ID cache had entries in it, all we have to do is check one out.
.. 380      1374 4
.. 381      1375 4
.. 382      1376 3      ELSE
```

```
383      1377      PMSSGL_FIDHIT = .PMSSGL_FIDHIT + 1;
384      1378
385      1379      FILE_NUMBER = .FID_CACHE[VCASL_FIDLIST];
386      1380      FID_CACHE[VCASW_FIDCOUNT] = .FID_CACHE[VCASW_FIDCOUNT] - 1;
387      1381      CHSMOVE (.FID_CACHE[VCASW_FIDCOUNT]*4,
388      1382              FID_CACHE[VCASL_FIDLIST]+4,
389      1383              FID_CACHE[VCASL_FIDLIST]);
390      1384
391      1385      NEW_FID = .FILE_NUMBER;
392      1386      NEW_FID_RVN = .CURRENT_RVN;          ! record for cleanup
393      1387
394      1388      ! Map the file header. If it fails to map, we have screwed up badly.
395      1389
396      1390
397      1391      VBN = .FILE_NUMBER + .VCB[VCBSB_IBMAPSIZE] + .VCB[VCBSW_CLUSTER]*4;
398      1392      LBN = MAP_IDX (.VBN);
399      1393      IF .LBN EQL -1 THEN BUG_CHECK (HDRNOTMAP, FATAL, 'Allocated file header not mapped');
400      1394
401      1395      FILE_ID[FIDSW_NUM] = .FILE_NUMBER<0,16>;
402      1396      FILE_ID[FIDSB_NMX] = .FILE_NUMBER<16,8>;
403      1397      FILE_ID[FIDSB_RVN] = .CURRENT_RVN;
404      1398
405      1399      ! If this is the creation of a new primary header, PRIM_LCKINDX will
406      1400      ! be zero. In that case, serialize further processing on that header.
407      1401      ! If extension headers are being allocated, the primary lock index has
408      1402      ! already been established.
409      1403
410      1404
411      1405      IF .PRIM_LCKINDX EQL 0
412      1406      THEN
413      1407      BEGIN
414      1408
415      1409      ! Release the allocation lock prior to serializing on this file id.
416      1410      ! This could be a valid header that another process is trying to modify
417      1411      ! allocation on, and if so, we would deadlock if the allocation lock
418      1412      ! were not released now.
419      1413
420      1414
421      1415      ALLOCATION_UNLOCK ();
422      1416      PRIM_LCKINDX = SERIAL_FILE (.FILE_ID);
423      1417      NEW_LCKINDX = 1;
424      1418      END;
425      1419
426      1420      ! Read the header; then check the block read for resemblance to a file header.
427      1421
428      1422
429      1423      HEADER = READ_NEW_HEADER (.LBN);
430      1424
431      1425      IF .HEADER NEQ 0
432      1426      THEN
433      1427      BEGIN
434      1428      FILE_ID[FIDSW_SEQ] = .HEADER[FH2SW_FID_SEQ];
435      1429      STATUS = CHECK_HEADER2 (.HEADER, .FILE_ID);
436      1430
437      1431      ! Make the final checks that the block is acceptable as a file header. We do
438      1432      ! not use valid file headers. Also, we skip file numbers with the low 16 bits
439      1433      ! all zero to avoid confusing the old FCS-11. Also skip file numbers in the
```

```

440 1434 4 ! reserved file number range to avoid total confusion if the volume is damaged.
441 1435 4 !
442 1436 4
443 1437 4 IF .FILE_ID[FID$W_NUM] EQL 0
444 1438 4 THEN
445 1439 4 WRITE_BLOCK (.HEADER)
446 1440 4 ELSE
447 1441 4 IF NOT .STATUS
448 1442 4 AND NOT (.FILE_ID[FID$B_NMX] EQL 0
449 1443 5 AND .FILE_ID[FID$W_NUM] LEQU .CURRENT_VCB[VCB$B_RESFILES])
450 1444 4 THEN EXITLOOP;
451 1445 4 END;
452 1446 4
453 1447 4 ! If we got this far, i.e., did not exit the loop, we do not want to use
454 1448 4 this file header for some reason. Before going around another time,
455 1449 4 release the serialization lock if we got one in this routine, and then
456 1450 4 reacquire the allocation lock for another pass around the loop.
457 1451 4
458 1452 4
459 1453 4 IF .NEW_LCKINDX
460 1454 4 THEN
461 1455 4 BEGIN
462 1456 4 IF .HEADER NEQ 0
463 1457 4 THEN INVALDATE (.HEADER);
464 1458 4 RELEASE_SERIAL_LOCK (.PRIM_LCKINDX);
465 1459 4 PRIM_LCKINDX = 0;
466 1460 4 ALLOCATION_LOCK ();
467 1461 4 END;
468 1462 4
469 1463 4 END; ! end of file number allocation loop
470 1464 4
471 1465 4 HEADER_LBN = .LBN; ! record LBN of new header
472 1466 4
473 1467 4 IF .STATUS EQL 0
474 1468 4 AND (.HEADER)<0,32> NEQ 0
475 1469 4 THEN FILE_ID[FID$W_SEQ] = .EXE$GQ SYSTIME<16,16>;
476 1470 4 FILE_ID[FID$W_SEQ] = .FILE_ID[FID$W_SEQ] + 1;
477 1471 4 CH$MOVE (FID$C_LENGTH, .FILE_ID, HEADER[FH2$W_FID]);
478 1472 4 HEADER[FH2$B_FID_RVN] = 0;
479 1473 4
480 1474 4 MARK_DIRTY (.HEADER);
481 1475 4 .HEADER
482 1476 4
483 1477 1 END; ! end of routine CREATE_HEADER

```

```

.TITLE CREHDR
.IDENT \V04-001\

.EXTRN CLU$GL_CLUB, PM$SGL_FIDHIT
.EXTRN PM$SGL_FIDMISS, EXE$GQ_SYSTIME
.EXTRN ALLOCATION_LOCK
.EXTRN ALLOCATION_UNLOCK
.EXTRN SERIAL_FILE, RELEASE_SERIAL_LOCK
.EXTRN DEQ_LOCK, READ_BLOCK
.EXTRN WRITE_BLOCK, DELETE_FID
.EXTRN RELEASE_LOCKBASIS

```


50

1C AB

0000G	CF		01	FB	000A1	CALLS	#1, RELEASE_LOCKBASIS	1311	
0000G	CF		00	FB	000A6	CALLS	#0, ALLOCATION_UNLOCK	1312	
	50	98	AA	DO	000AB	MOVL	-104(BASE), RO		
	50	0E	A0	3C	000AF	MOVZWL	14(RO), RO		
	50		18	78	000B3	ASHL	#24, RO, RO		
	50		01	88	000B7	BISB2	#1, IDX_FILE_ID		
		24	AE	D4	000BA	CLRL	LOCK_ID	1313	
			01	DD	000BD	PUSHL	#1	1314	
		28	AE	9F	000BF	PUSHAB	LOCK_ID		
			50	DD	000C2	PUSHL	IDX_FILE_ID		
0000G	CF		03	FB	000C4	CALLS	#3, CACHE_LOCK	1315	
0000G	CF		00	FB	000C9	CALLS	#0, ALLOCATION_LOCK	1316	
		24	AE	DD	000CE	PUSHL	LOCK_ID		
0000G	CF		01	FB	000D1	CALLS	#1, DEQ_LOCK	1317	
1C	AE		01	CE	000D6	MNEGL	#1, CACHE_FLUSHED	1318	
			05	11	000DA	BRB	9\$	1321	
0000G	CF		00	FB	000DC	CALLS	#0, EXTEND_INDEX	1300	
			FF34	31	000E1	BRW	1\$	1334	
04	AE	14	AA	DO	000E4	MOVL	20(BASE), TEMP	1335	
		24	AB	9F	000E9	PUSHAB	36(IDX_FCB)		
0000G	CF		01	FB	000EC	CALLS	#1, SERIAL_FILE	1337	
		28	AE	9F	000F1	PUSHAB	COUNT		
		01	A7	9F	000F4	PUSHAB	1(CURRENT_EOF)		
0000V	CF		02	FB	000F7	CALLS	#2, MAP_IDX		
10	AE		50	DO	000FC	MOVL	RO, LBN		
		FF78	CA	DD	00100	PUSHL	-136(BASE)	1338	
		2C	AE	DD	00104	PUSHL	COUNT		
		18	AE	DD	00107	PUSHL	LBN		
0000G	CF		03	FB	0010A	CALLS	#3, ERASE_BLOCKS	1339	
	57	28	AE	CO	0010F	ADDL2	COUNT, CURRENT_EOF	1341	
0000V	CF		00	FB	00113	CALLS	#0, READ_IDX_HEADER		
	58		50	DO	00118	MOVL	RO, HEADER		
	50	01	A7	9E	0011B	MOVAB	1(R7), RO	1342	
	50		10	9C	0011F	ROTL	#16, RO, 28(HEADER)		
		20	A8	B4	00124	CLRW	32(HEADER)	1343	
			68	91	00127	CMPB	(HEADER), #40	1344	
			05	1F	0012A	BLSSU	11\$		
4C	AB	01	A7	9E	0012C	MOVAB	1(R7), 76(HEADER)	1345	
			58	DD	00131	PUSHL	HEADER	1347	
0000G	CF		01	FB	00133	CALLS	#1, CHECKSUM		
0000G	CF		00	FB	00138	CALLS	#0, WRITE_HEADER	1348	
3C	AB		57	DO	0013D	MOVL	CURRENT_EOF, 60(IDX_FCB)	1349	
		2C	A9	DD	00141	PUSHL	44(VCB)	1350	
			58	DD	00144	PUSHL	HEADER		
0000G	CF		02	FB	00146	CALLS	#2, RESET_LBN		
			58	DD	0014B	PUSHL	HEADER	1351	
0000G	CF		01	FB	0014D	CALLS	#1, WRITE_BLOCK		
			58	DD	00152	PUSHL	HEADER	1352	
0000G	CF		01	FB	00154	CALLS	#1, INVALIDATE		
		14	AA	DD	00159	PUSHL	20(BASE)	1354	
0000G	CF		01	FB	0015C	CALLS	#1, RELEASE_SERIAL_LOCK		
14	AA		04	AE	DO	00161	MOVL	TEMP, 20(BASE)	1355
			FEAF	31	00166	BRW	1\$	1361	
		0C	AE	DD	00169	PUSHL	BUFFER	1369	
0000G	CF		01	FB	0016C	CALLS	#1, WRITE_BLOCK		
			06	11	00171	BRB	14\$	1265	
		00000000G	00	D6	00173	INCL	PMSSGL_FIDHIT	1377	

	14	AE	24	A6	DO	00179	14\$:	MOVL	36(FID_CACHE), FILE_NUMBER	1379
			02	A6	B7	0017E		DECW	2(FID_CACHE)	1380
		50	02	A6	3C	00181		MOVZWL	2(FID_CACHE), R0	1381
		50		04	C4	00185		MULL2	#4, R0	
24	A6	28		50	28	00188		MOVCS	R0, 40(FID_CACHE), 36(FID_CACHE)	1383
		A8		AA	DO	0018E		MOVL	FILE_NUMBER, -88(BASE)	1385
		AC		AA	DO	00193		MOVL	-96(BASE), -84(BASE)	1386
				50	A9	9A	00198	MOVZBL	56(VCB), R0	1391
	51			50	AE	C1	0019C	ADDL3	FILE_NUMBER, R0, R1	
				50	A9	3C	001A1	MOVZWL	60(VCB), R0	
		18		AE	6140	DE	001A5	MOVAL	(R1)[R0], VBN	
					AE	DD	001AA	PUSHL	VBN	1392
	0000V	CF		01	FB	001AD		CALLS	#1, MAP_IDX	
	10	AE		50	DO	001B2		MOVL	R0, LBN	
	FFFFFFF	8F		10	AE	D1	001B6	CMPL	LBN, #-1	1393
					04	12	001BE	BNEQ	15\$	
						FEFF	001C0	BUGW		
						0000+	001C2	.WORD	<BUGS HDRNOTMAP!4>	
	04	BC	14	AE	B0	001C4	15\$:	MOVW	FILE_NUMBER, @FILE_ID	1395
		50	04	AC	DO	001C9		MOVL	FILE_ID, R0	1396
	05	A0	16	AE	90	001CD		MOVW	FILE_NUMBER+2, 5(R0)	
		50	04	AC	DO	001D2		MOVL	FILE_ID, R0	1397
	04	A0	A0	AA	90	001D6		MOVW	-96(BASE), 4(R0)	
			18	AA	D5	001DB		TSTL	24(BASE)	1405
				15	12	001DE		BNEQ	16\$	
	0000G	CF		00	FB	001E0		CALLS	#0, ALLOCATION_UNLOCK	1415
			04	AC	DD	001E5		PUSHL	FILE_ID	1416
	0000G	CF		01	FB	001E8		CALLS	#1, SERIAL_FILE	
	18	AA		50	DO	001ED		MOVL	R0, 24(BASE)	
	20	AE		01	DO	001F1		MOVL	#1, NEW_LCKINDX	1417
			10	AE	DD	001F5	16\$:	PUSHL	LBN	1423
	0000V	CF		01	FB	001F8		CALLS	#1, READ_NEW_HEADER	
		58		50	DO	001FD		MOVL	R0, HEADER	
				3E	13	00200		BEQL	18\$	1425
		50	04	AC	DO	00202		MOVL	FILE_ID, R0	1428
	02	A0	0A	A8	B0	00206		MOVW	10(HEADER), 2(R0)	
			04	AC	DD	0020B		PUSHL	FILE_ID	1429
				58	DD	0020E		PUSHL	HEADER	
	0000G	CF		02	FB	00210		CALLS	#2, CHECK_HEADER2	
	08	AE		50	DO	00215		MOVL	R0, STATUS	
		52	04	AC	DO	00219		MOVL	FILE_ID, R2	1437
				62	B5	0021D		TSTW	(R2)	
				09	12	0021F		BNEQ	17\$	
				58	DD	00221		PUSHL	HEADER	1439
	0000G	CF		01	FB	00223		CALLS	#1, WRITE_BLOCK	
				16	11	00228		BRB	18\$	
		12	08	AE	E8	0022A	17\$:	BLBS	STATUS, 18\$	1441
			05	A2	95	0022E		TSTB	5(R2)	1442
				2F	12	00231		BNEQ	21\$	
		50	98	AA	DO	00233		MOVL	-104(BASE), R0	1443
		51	4F	A0	9A	00237		MOVZBL	79(R0), R1	
		62		51	B1	0023B		CMPW	R1, (R2)	
				22	1F	0023E		BLSSU	21\$	
		1B	20	AE	E9	00240	18\$:	BLBC	NEW_LCKINDX, 20\$	1453
				58	D5	00244		TSTL	HEADER	1456
				07	13	00246		BEQL	19\$	
				58	DD	00248		PUSHL	HEADER	1457

0000G	CF		01	FB	0024A		CALLS	#1, INVALIDATE	:	
		18	AA	DD	0024F	19\$:	PUSHL	24(BASE)	:	1458
0000G	CF		01	FB	00252		CALLS	#1, RELEASE_SERIAL_LOCK	:	
		18	AA	D4	00257		CLRL	24(BASE)	:	1459
0000G	CF		00	FB	0025A		CALLS	#0, ALLOCATION_LOCK	:	1460
			FDB6	31	0025F	20\$:	BRW	1\$:	1256
B0	AA		10	AE	D0	21\$:	MOVL	LBN, -80(BASE)	:	1465
			08	AE	D5		TSTL	STATUS	:	1467
			10	12	0026A		BNEQ	22\$:	
			68	D5	0026C		TSTL	(HEADER)	:	1468
			0C	13	0026E		BEQL	22\$:	
	50		04	AC	D0		MOVL	FILE_ID, R0	:	1469
02	A0	00000000G	00	B0	00274		MOVW	EXESGQ, SYSTIME+2, 2(R0)	:	
	50		04	AC	D0	22\$:	MOVL	FILE_ID, R0	:	1470
			02	A0	B6	00280	INCL	2(R0)	:	
08	A8		04	BC	06		MOVCL	#6, @FILE_ID, 8(HEADER)	:	1471
			0C	A8	94	00289	CLRB	12(HEADER)	:	1472
				58	DD	0028C	PUSHL	HEADER	:	1474
0000G	CF		01	FB	0028E		CALLS	#1, MARK_DIRTY	:	
	50		58	D0	00292		MOVL	HEADER, R0	:	1477
				04	00296		RET		:	

; Routine Size: 663 bytes, Routine Base: \$CODE\$ + 0000

```

485 1478 1 ROUTINE FILL_FID_CACHE (VCB, BUFFER, VBN) : L_NORM NOVALUE =
486 1479 1
487 1480 1 ++
488 1481 1
489 1482 1 FUNCTIONAL DESCRIPTION:
490 1483 1
491 1484 1 This routine refills the cache from the supplied bitmap buffer.
492 1485 1 It will not fill the cache with file ID's that represent
493 1486 1 headers past the current index file EOF.
494 1487 1
495 1488 1
496 1489 1 CALLING SEQUENCE:
497 1490 1 FILL_FID_CACHE (ARG1, ARG2, ARG3)
498 1491 1
499 1492 1 INPUT PARAMETERS:
500 1493 1 ARG1: address of volume VCB
501 1494 1 ARG2: address of bitmap buffer
502 1495 1 ARG3: relative block number in bitmap
503 1496 1
504 1497 1 IMPLICIT INPUTS:
505 1498 1 NONE
506 1499 1
507 1500 1 OUTPUT PARAMETERS:
508 1501 1 NONE
509 1502 1
510 1503 1 IMPLICIT OUTPUTS:
511 1504 1 NONE
512 1505 1
513 1506 1 ROUTINE VALUE:
514 1507 1 NONE
515 1508 1
516 1509 1 SIDE EFFECTS:
517 1510 1 file ID cache modified
518 1511 1
519 1512 1 --
520 1513 1
521 1514 2 BEGIN
522 1515 2
523 1516 2 MAP
524 1517 2 VCB : REF BBLOCK, ! local copy of VCB address
525 1518 2 BUFFER : REF BITVECTOR; ! address of index file bitmap buffer
526 1519 2
527 1520 2 LOCAL
528 1521 2 CACHE : REF BBLOCK, ! pointer to cache block
529 1522 2 FID_CACHE : REF BBLOCK, ! pointer to file ID cache
530 1523 2 ADDRESS : REF BITVECTOR, ! address of byte in buffer
531 1524 2 FREE_COUNT, ! count of cache entries to fill
532 1525 2 BITPOS, ! bit position of free bit within byte
533 1526 2 BITPOS2, ! bit position of first used bit
534 1527 2 FILE_NUMBER, ! file number found
535 1528 2 IDX_VBN; ! current block in index bitmap
536 1529 2
537 1530 2 BIND_COMMON;
538 1531 2
539 1532 2
540 1533 2 ! If the cache is not currently marked valid, attempt to take out the
541 1534 2 ! cache lock if we are in a cluster and may do so.

```



```

: 542      1535      2 :
: 543      1536      2 :
: 544      1537      2 CACHE = .VCB[VCBSL_CACHE];
: 545      1538      2 FID_CACHE = .CACHE[VCSL_FIDCACHE];
: 546      1539      2 IF NOT .CACHE[VCSV_FIDC_VALID]
: 547      1540      2 THEN INIT_FID_CACHE(.CACHE);
: 548      1541      2 :
: 549      1542      2 ! Fill the cache from the supplied bitmap buffer. Find each byte containing
: 550      1543      2 ! a free bit, and then find the free bit.
: 551      1544      2 :
: 552      1545      2 :
: 553      1546      2 ADDRESS = .BUFFER;
: 554      1547      2 FREE_COUNT = .FID_CACHE[VCSW_FIDSIZE]/2 - .FID_CACHE[VCSW_FIDCOUNT] + 1;
: 555      1548      2 :
: 556      1549      2 WHILE 1 DO
: 557      1550      2 BEGIN
: 558      1551      2 IF CH$FAIL (ADDRESS = CH$FIND_NOT_CH (.BUFFER+512-.ADDRESS, .ADDRESS, 255))
: 559      1552      2 THEN EXITLOOP;
: 560      1553      2 FFC (%REF (0), %REF (8), .ADDRESS, BITPOS);
: 561      1554      2 FILE_NUMBER = .VBN*4096 + (.ADDRESS-.BUFFER)*8 + .BITPOS + 1;
: 562      1555      2 :
: 563      1556      2 ! Check file number against index file EOF and the maximum file limit.
: 564      1557      2 :
: 565      1558      2 :
: 566      1559      2 IF .FILE_NUMBER + .VCB[VCBSB_IBMAPSIZE] + .VCB[VCSW_CLUSTER]*4
: 567      1560      2 GTRU .BBLOCK [.VCB[VCBSL_FCBFL], FCB$E_FBLK]
: 568      1561      2 OR .FILE_NUMBER GTRU .VCB[VCBSL_MAXFILES]
: 569      1562      2 THEN EXITLOOP;
: 570      1563      2 :
: 571      1564      2 ! Enter the file number in the cache and mark it busy in the bitmap.
: 572      1565      2 ! Exit the loop if the cache is now full enough.
: 573      1566      2 :
: 574      1567      2 :
: 575      1568      2 ADDRESS[.BITPOS] = 1;
: 576      1569      2 FID_CACHE[VCSW_FIDCOUNT] = .FID_CACHE[VCSW_FIDCOUNT] + 1;
: 577      1570      2 VECTOR [FID_CACHE[VCSL_FIDLIST], .FID_CACHE[VCSW_FIDCOUNT]-1] = .FILE_NUMBER;
: 578      1571      2 FREE_COUNT = .FREE_COUNT - 1;
: 579      1572      2 IF .FREE_COUNT LEQ 0
: 580      1573      2 OR NOT .CACHE[VCSV_FIDC_VALID]
: 581      1574      2 THEN EXITLOOP;
: 582      1575      2 END;                                     ! end of bitmap processing loop
: 583      1576      2 :
: 584      1577      2 IDX_VBN = .VBN;                           ! update current VBN of index file bitmap
: 585      1578      2 IF .FILE_NUMBER<0,12> EQL 0
: 586      1579      2 THEN IDX_VBN = .IDX_VBN + 1;
: 587      1580      2 VCB[VCBSB_IBMAPVBN] = .IDX_VBN;
: 588      1581      2 :
: 589      1582      1 END;                                     ! end of routine FILL_FID_CACHE

```

01FC 0000 FILL_FID_CACHE:

50	04	AC	DO	00002	.WORD	Save R2,R3,R4,R5,R6,R7,R8	: 1478
54	58	AO	DO	00006	MOVL	VCB, R0	: 1537
					MOVL	88(R0), CACHE	:

		52		64	D0	0000A	MOVL	(CACHE), FID_CACHE	1538
		07	0B	A4	E8	0000D	BLBS	11(CACHE), 1\$	1539
				54	DD	00011	PUSHL	CACHE	1540
	0000V	CF		01	FB	00013	CALLS	#1, INIT FID_CACHE	
		57	08	AC	D0	00018	MOVL	BUFFER, ADDRESS	1546
		53		62	3C	0001C	MOVZWL	(FID_CACHE), R3	1547
		53		02	C6	0001F	DIVL2	#2, R3	
		50	02	A2	3C	00022	MOVZWL	2(FID_CACHE), R0	
		53		50	C2	00026	SUBL2	R0, R3	
				53	D6	00029	INCL	FREE_COUNT	
50	08	AC		57	C3	0002B	SUBL3	ADDRESS, BUFFER, R0	1551
		50	0200	C0	9E	00030	MOVAB	512(R0), R0	
67		50	FF	8F	3B	00035	SKPC	#255, R0, (ADDRESS)	
				02	12	0003A	BNEQ	3\$	
				51	D4	0003C	CLRL	R1	
		57		51	D0	0003E	MOVL	R1, ADDRESS	
				53	13	00041	BEQL	5\$	
58	67	08		00	EB	00043	FFC	#0, #8, (ADDRESS), BITPOS	1553
	50	AC	0C	0C	78	00048	ASHL	#12, VBN, R0	1554
	51	57	08	AC	C3	0004D	SUBL3	BUFFER, ADDRESS, R1	
		50		6041	7E	00052	MOVAQ	(R0)[R1], R0	
		56	01	A840	9E	00056	MOVAB	1(BITPOS)[R0], FILE_NUMBER	
		51	04	AC	D0	0005B	MOVL	VCB, R1	1559
		50	38	A1	9A	0005F	MOVZBL	56(R1), R0	
55		56		50	C1	00063	ADDL3	R0, FILE_NUMBER, R5	
		50	3C	A1	3C	00067	MOVZWL	60(R1), R0	
		55		6540	DE	0006B	MOVAL	(R5)[R0], R5	
		50		61	D0	0006F	MOVL	(R1), R0	1560
	3C	A0		55	D1	00072	CMPL	R5, 60(R0)	
				1E	1A	00076	BGTRU	5\$	
	44	A1		56	D1	00078	CMPL	FILE_NUMBER, 68(R1)	1561
				18	1A	0007C	BGTRU	5\$	
00		67		58	E2	0007E	BBSS	BITPOS, (ADDRESS), 4\$	1568
			02	A2	B6	00082	INCL	2(FID_CACHE)	1569
		50	02	A2	3C	00085	MOVZWL	2(FID_CACHE), R0	1570
	20	A240		56	D0	00089	MOVL	FILE_NUMBER, 32(FID_CACHE)[R0]	
				53	D7	0008E	DECL	FREE_COUNT	1571
				04	15	00090	BLEQ	5\$	1572
		95	0B	A4	E8	00092	BLBS	11(CACHE), 2\$	1573
		51	0C	AC	D0	00096	MCVL	VBN, IDX_VBN	1577
	OFFF	8F		56	B3	0009A	BITW	FILE_NUMBER, #4095	1578
				02	12	0009F	BNEQ	6\$	
				51	D6	000A1	INCL	IDX_VBN	1579
	3A	50	04	AC	D0	000A3	MOVL	VCB, R0	1580
		A0		51	90	000A7	MOVB	IDX_VBN, 58(R0)	
				04	00	000AB	RET		1582

; Routine Size: 172 bytes, Routine Base: \$CODE\$ + 0297

```

: 591 1583 1 GLOBAL ROUTINE INIT_FID_CACHE (CACHE) : L_NORM NOVALUE =
: 592 1584 1
: 593 1585 1 !++
: 594 1586 1
: 595 1587 1 FUNCTIONAL DESCRIPTION:
: 596 1588 1
: 597 1589 1 This routine refills the cache from the supplied bitmap buffer.
: 598 1590 1 It will not fill the cache with file ID's that represent
: 599 1591 1 headers past the current index file EOF.
: 600 1592 1
: 601 1593 1
: 602 1594 1 CALLING SEQUENCE:
: 603 1595 1 INIT_FID_CACHE (CACHE)
: 604 1596 1
: 605 1597 1 INPUT PARAMETERS:
: 606 1598 1 CACHE: pointer to main cache block
: 607 1599 1
: 608 1600 1 IMPLICIT INPUTS:
: 609 1601 1 NONE
: 610 1602 1
: 611 1603 1 OUTPUT PARAMETERS:
: 612 1604 1 NONE
: 613 1605 1
: 614 1606 1 IMPLICIT OUTPUTS:
: 615 1607 1 NONE
: 616 1608 1
: 617 1609 1 ROUTINE VALUE:
: 618 1610 1 NONE
: 619 1611 1
: 620 1612 1 SIDE EFFECTS:
: 621 1613 1 cache marked valid, lock taken out
: 622 1614 1
: 623 1615 1 --
: 624 1616 1
: 625 1617 2 BEGIN
: 626 1618 2
: 627 1619 2 MAP
: 628 1620 2 CACHE : REF BBLOCK; ! pointer to cache block
: 629 1621 2
: 630 1622 2 LOCAL
: 631 1623 2 FID_CACHE : REF BBLOCK, ! pointer to file ID cache
: 632 1624 2 INDEX_FID; ! lock basis for index file
: 633 1625 2
: 634 1626 2 BIND_COMMON;
: 635 1627 2
: 636 1628 2 EXTERNAL
: 637 1629 2 CLUSGL_CLUB : ADDRESSING_MODE (GENERAL);
: 638 1630 2
: 639 1631 2 EXTERNAL ROUTINE
: 640 1632 2 CACHE_LOCK : L_NORM; ! acquire special cache lock
: 641 1633 2
: 642 1634 2
: 643 1635 2 ! If the cache is not currently marked valid, attempt to take out the
: 644 1636 2 cache lock if we are in a cluster and may do so.
: 645 1637 2
: 646 1638 2
: 647 1639 2 FID_CACHE = .CACHE[VCASL_FIDCACHE];

```

```

: 645      1640 2 IF NOT .BBLOCK [CURRENT UCBC[UCBSL_DEVCHAR], DEVSV_DMT]
: 646      1641 2 AND NOT .CURRENT VCB[VCBSV_WRITE_IF]
: 647      1642 2 AND .FID_CACHE[VCASW_FIDSIZE] GTRU 1
: 648      1643 2 THEN
: 649      1644 2 BEGIN
: 650      1645 2 IF .BBLOCK [CURRENT UCBC[UCBSL_DEVCHAR2], DEVSV_CLU]
: 001 CDS0018 1646 2 AND .CLUSGL_CLUB NEQ 0
: 651      1647 2 THEN
: 652      1648 2 BEGIN
: 653      1649 2 INDEX_FID = FIDSC_INDEXF OR .CURRENT VCB[VCBSW_RVN] ^ 24;
: 654      1650 2 IF CACHE_LOCK (.INDEX_FID, FID_CACHE[VCASL_FID[CLKID], 0)
: 655      1651 2 THEN CACHE[VCASV_FIDC_VALID] = -1;
: 656      1652 2 END
: 657      1653 2 ELSE
: 658      1654 2 CACHE[VCASV_FIDC_VALID] = 1;
: 659      1655 2 END;
: 660      1656 2
: 661      1657 1 END;

```

! end of routine INIT_FID_CACHE

			000C 00000	.ENTRY	INIT_FID_CACHE, Save R2,R3	: 1583
	52	04	AC D0 00002	MOVL	CACHE, R2	: 1639
	53		62 D0 00006	MOVL	(R2), FID_CACHE	
	51	94	AA D0 00009	MOVL	-108(BASE), R1	: 1640
44	3A	A1	05 E0 0000D	BBS	#5, 58(R1), 2\$	
	50	98	AA D0 00012	MOVL	-104(BASE), R0	: 1641
	3C	0B	A0 E8 00016	BLBS	11(R0), 2\$	
	01		63 B1 0001A	CMPL	(FID_CACHE), #1	: 1642
			37 1B 0001D	BLEQU	2\$	
	2F	3C	A1 E9 0001F	BLBC	60(R1), 1\$: 1645
		00000000G	00 D5 00023	TSTL	CLUSGL_CLUB	: 1646
			27 13 00029	BEQL	1\$	
	50	98	AA D0 0002B	MOVL	-104(BASE), R0	: 1649
	50	0E	A0 3C 0002F	MOVZWL	14(R0), R0	
50			18 78 00033	ASHL	#24, R0, R0	
	50		01 88 00037	BISB2	#1, INDEX_FID	
			7E D4 0003A	CLRL	-(SP)	: 1650
		04	A3 9F 0003C	PUSHAB	4(FID_CACHE)	
			50 DD 0003F	PUSHL	INDEX_FID	
	0000G	CF	03 FB 00041	CALLS	#3, CACHE_LOCK	
		0D	50 E9 00046	BLBC	R0, 2\$	
	50	04	AC D0 00049	MOVL	CACHE, R0	: 1651
	0B	A0	01 88 0004D	BISB2	#1, 11(R0)	
			04 00051	RET		: 1645
	0B	A2	01 88 00052 1\$:	BISB2	#1, 11(R2)	: 1654
			04 00056 2\$:	RET		: 1657

; Routine Size: 87 bytes, Routine Base: \$CODE\$ + 0343

```

: 663      1658 1 ROUTINE READ_NEW_HEADER (LBN) : L_NORM =
: 664      1659 1
: 665      1660 1 !++
: 666      1661 1
: 667      1662 1 FUNCTIONAL DESCRIPTION:
: 668      1663 1
: 669      1664 1     This routine reads the block about to be used for a new file header.
: 670      1665 1     It uses a local condition handler to fix up errors.
: 671      1666 1
: 672      1667 1
: 673      1668 1 CALLING SEQUENCE:
: 674      1669 1     READ_NEW_HEADER (ARG1)
: 675      1670 1
: 676      1671 1 INPUT PARAMETERS:
: 677      1672 1     ARG1: LBN of block to read
: 678      1673 1
: 679      1674 1 IMPLICIT INPUTS:
: 680      1675 1     NONE
: 681      1676 1
: 682      1677 1 OUTPUT PARAMETERS:
: 683      1678 1     NONE
: 684      1679 1
: 685      1680 1 IMPLICIT OUTPUTS:
: 686      1681 1     NONE
: 687      1682 1
: 688      1683 1 ROUTINE VALUE:
: 689      1684 1     address of buffer containing block or 0 if bad
: 690      1685 1
: 691      1686 1 SIDE EFFECTS:
: 692      1687 1     block read and/or written
: 693      1688 1
: 694      1689 1 !--
: 695      1690 1
: 696      1691 2 BEGIN
: 697      1692 2
: 698      1693 2 LOCAL
: 699      1694 2     HEADER          : REF BBLOCK;    ! address of block read
: 700      1695 2
: 701      1696 2 BASE_REGISTER;
: 702      1697 2
: 703      1698 2 EXTERNAL ROUTINE
: 704      1699 2     READ_BLOCK      : L_NORM,        ! read a block
: 705      1700 2     WRITE_BLOCK     : L_NORM,        ! write a block
: 706      1701 2     INVALIDATE     : L_NORM,        ! invalidate a buffer
: 707      1702 2     CREATE_BLOCK    : L_NORM;        ! create a new block buffer
: 708      1703 2
: 709      1704 2 ! Under control of the condition handler, we read the block. If the read
: 710      1705 2 ! fails, we attempt to rewrite the block and then read it again. If either
: 711      1706 2 ! of the latter fails, we return failure.
: 712      1707 2
: 713      1708 2
: 714      1709 2 ENABLE HANDLER;
: 715      1710 2
: 716      1711 2 HEADER = READ_BLOCK (.LBN, 1, HEADER_TYPE);
: 717      1712 2
: 718      1713 2 IF .HEADER EQL 0
: 719      1714 2 THEN

```

```

: 720      1715 3 BEGIN
: 721      1716 3 HEADER = CREATE_BLOCK (.LBN, 1, HEADER_TYPE);
: 722      1717 3 (.HEADER)<0,32> = 1;
: 723      1718 3 WRITE_BLOCK (.HEADER);
: 724      1719 3 INVALIDATE (.HEADER);
: 725      1720 3 HEADER = READ_BLOCK (.LBN, 1, HEADER_TYPE);
: 726      1721 3 END;
: 727      1722 3
: 728      1723 2 RETURN .HEADER;
: 729      1724 2
: 730      1725 1 END;
! end of routine READ_NEW_HEADER

```

```

0004 00000 READ_NEW_HEADER:
      6D 0042 CF DE 00002 .WORD Save R2
      7E 01 7D 00007 MOVAL 2$, (FP)
      04 AC DD 0000A MOVQ #1, -(SP)
0000G CF 03 FB 0000D PUSHL LBN
      52 50 D0 00012 CALLS #3, READ_BLOCK
      7E 2D 12 00015 MOVL R0, HEADER
      04 01 7D 00017 BNEQ 1$
      04 AC DD 0001A MOVQ #1, -(SP)
0000G CF 03 FB 0001D PUSHL LBN
      52 50 D0 00022 CALLS #3, CREATE_BLOCK
      62 01 D0 00025 MOVL #1, (HEADER)
0000G CF 52 DD 00028 PUSHL HEADER
      01 FB 0002A CALLS #1, WRITE_BLOCK
0000G CF 52 DD 0002F PUSHL HEADER
      7E 01 FB 00031 CALLS #1, INVALIDATE
      04 AC DD 00039 MOVQ #1, -(SP)
0000G CF 03 FB 0003C PUSHL LBN
      52 50 D0 00041 CALLS #3, READ_BLOCK
      50 52 D0 00044 MOVL R0, HEADER
      04 00047 MOVL HEADER, R0
      0000 00048 1$: RET
      7E D4 0004A .WORD Save nothing
      5E DD 0004C CLRL -(SP)
0000V 7E 04 AC 7D 0004E PUSHL SP
      CF 03 FB 00052 MOVQ 4(AP), -(SP)
      04 00057 CALLS #3, HANDLER
      RET
      : 1658
      : 1691
      : 1711
      : 1713
      : 1716
      : 1717
      : 1718
      : 1719
      : 1720
      : 1723
      : 1725
      : 1691

```

; Routine Size: 88 bytes, Routine Base: \$CODE\$ + 039A

```

: 732      1726 1 ROUTINE HANDLER (SIGNAL, MECHANISM) =
: 733      1727 1
: 734      1728 1 !++
: 735      1729 1
: 736      1730 1 FUNCTIONAL DESCRIPTION:
: 737      1731 1
: 738      1732 1     This routine is the condition handler for the initial header read.
: 739      1733 1     On surface errors, it unwinds and causes a return of 0 to the caller
: 740      1734 1     of the I/O routine to indicate error. Hard drive errors cause the
: 741      1735 1     usual error exit.
: 742      1736 1
: 743      1737 1 CALLING SEQUENCE:
: 744      1738 1     HANDLER (ARG1, ARG2)
: 745      1739 1
: 746      1740 1 INPUT PARAMETERS:
: 747      1741 1     ARG1: address of signal array
: 748      1742 1     ARG2: address of mechanism array
: 749      1743 1
: 750      1744 1 IMPLICIT INPUTS:
: 751      1745 1     NONE
: 752      1746 1
: 753      1747 1 OUTPUT PARAMETERS:
: 754      1748 1     NONE
: 755      1749 1
: 756      1750 1 IMPLICIT OUTPUTS:
: 757      1751 1     NONE
: 758      1752 1
: 759      1753 1 ROUTINE VALUE:
: 760      1754 1     $$$_RESIGNAL or none if unwind
: 761      1755 1
: 762      1756 1 SIDE EFFECTS:
: 763      1757 1     NONE
: 764      1758 1
: 765      1759 1 --
: 766      1760 1
: 767      1761 1
: 768      1762 2 BEGIN
: 769      1763 2
: 770      1764 2 MAP
: 771      1765 2     SIGNAL          : REF BBLOCK,   ! signal arg array
: 772      1766 2     MECHANISM       : REF BBLOCK;  ! mechanism arg array
: 773      1767 2
: 774      1768 2
: 775      1769 2 ! If the condition is change mode to user (error exit) and the status is
: 776      1770 2 ! read error, zero the return R0 and unwind to the the establisher. On
: 777      1771 2 ! most write errors, zero the return R0 and unwind to the caller.
: 778      1772 2 ! Otherwise, just resignal the condition.
: 779      1773 2 !
: 780      1774 2
: 781      1775 2 IF .SIGNAL[CHFSL_SIG_NAME] EQL $$$_CMODUSER
: 782      1776 2 THEN
: 783      1777 2     BEGIN
: 784      1778 2     MECHANISM[CHFSL_MCH_SAVRO] = 0;
: 785      1779 2
: 786      1780 2     IF SURFACE_ERROR (.SIGNAL[CHFSL_SIG_ARG1])
: 787      1781 2     THEN
: 788      1782 4     SUNWIND (DEPADR = MECHANISM[CHFSL_MCH_DEPTH])

```

```

: 789      1783 2      END;
: 790      1784 2
: 791      1785 2 RETURN SSS_RESIGNAL;
: 792      1786 2
: 793      1787 1 END;

```

```

! status is irrelevant if unwinding
! end of routine HANDLER

```

.EXTRN SYSSUNWIND

			0000 0000	HANDLER: .WORD	Save nothing	
	50	04	AC D0 00002	MOVL	SIGNAL, R0	: 1726
00000424	8F	04	A0 D1 00006	CPL	4(R0), #1060	: 1775
			41 12 0000E	BNEQ	2\$	
	50	08	AC D0 00010	MOVL	MECHANISM, R0	: 1778
		0C	A0 D4 00014	CLRL	12(R0)	
	50	04	AC D0 00017	MOVL	SIGNAL, R0	: 1780
000001F4	8F	08	A0 D1 0001B	CPL	8(R0), #500	
			1E 13 00023	BEQL	1\$	
0000005C	8F	08	A0 D1 00025	CPL	8(R0), #92	
			14 13 0002D	BEQL	1\$	
000000BC	8F	08	A0 D1 0002F	CPL	8(R0), #188	
			0A 13 00037	BEQL	1\$	
00002144	8F	08	A0 D1 00039	CPL	8(R0), #8516	
			0E 12 00041	BNEQ	2\$	
			7E D4 00043	CLRL	-(SP)	: 1782
7E	08	AC	08 C1 00045	ADDL3	#8, MECHANISM, -(SP)	
00000000G	00		02 FB 0004A	CALLS	#2, SYSSUNWIND	
	50	0918	8F 3C 00051	MOVZWL	#2328, R0	: 1785
			04 00056	RET		: 1787

; Routine Size: 87 bytes, Routine Base: \$CODE\$ + 03F2


```

: 795      1788 1 GLOBAL ROUTINE READ_IDX_HEADER : L_NORM =
: 796      1789 1
: 797      1790 1 |++
: 798      1791 1
: 799      1792 1 | FUNCTIONAL DESCRIPTION:
: 800      1793 1 |
: 801      1794 1 |     This routine reads the volume's index file header, using the
: 802      1795 1 |     alternate if it seems appropriate.
: 803      1796 1 |
: 804      1797 1 | CALLING SEQUENCE:
: 805      1798 1 |     READ_IDX_HEADER ()
: 806      1799 1 |
: 807      1800 1 | INPUT PARAMETERS:
: 808      1801 1 |     NONE
: 809      1802 1 |
: 810      1803 1 | IMPLICIT INPUTS:
: 811      1804 1 |     CURRENT_VCB: VCB of volume
: 812      1805 1 |
: 813      1806 1 | OUTPUT PARAMETERS:
: 814      1807 1 |     NONE
: 815      1808 1 |
: 816      1809 1 | IMPLICIT OUTPUTS:
: 817      1810 1 |     NONE
: 818      1811 1 |
: 819      1812 1 | ROUTINE VALUE:
: 820      1813 1 |     address of file header read
: 821      1814 1 |
: 822      1815 1 | SIDE EFFECTS:
: 823      1816 1 |     NONE
: 824      1817 1 |
: 825      1818 1 | --
: 826      1819 1
: 827      1820 2 BEGIN
: 828      1821 2
: 829      1822 2
: 830      1823 2 LOCAL
: 831      1824 2     HEADER          : REF BBLOCK,  ! address of header read
: 832      1825 2     FCB          : REF BBLOCK;  ! address of index file FCB
: 833      1826 2
: 834      1827 2 BIND_COMMON;
: 835      1828 2
: 836      1829 2 EXTERNAL ROUTINE
: 837      1830 2     FILE_SIZE      : L_NORM,      ! compute file header file size
: 838      1831 2     READ_HEADER    : L_NORM,      ! read file header
: 839      1832 2     READ_BLOCK     : L_NORM,      ! read a disk block
: 840      1833 2     CHECK_HEADER2  : L_NORM,      ! validate file header
: 841      1834 2     RESET_LBN    : L_NORM,      ! reassign LBN of buffer
: 842      1835 2     INVALIDATE   : L_NORM;      ! invalidate buffer
: 843      1836 2
: 844      1837 2
: 845      1838 2 ! Read the index file header. Check the file size against the
: 846      1839 2 ! file size in the FCB. A mismatch indicates a failure in writing the
: 847      1840 2 ! header the last time; if this occurs, try the alternate header instead.
: 848      1841 2
: 849      1842 2
: 850      1843 2 SAVE_STATUS = .USER_STATUS;
: 851      1844 2

```

```

: 852      1845 2 FCB = .CURRENT VCB[VCBSL_FCBFL];
: 853      1846 2 HEADER = READ HEADER (0, .FCB);
: 854      1847 2 IF FILE_SIZE T.HEADER) LSSU .FCB[FCBSL_FILESIZE]
: 855      1848 2 THEN
: 856      1849 2 BEGIN
: 857      1850 2 FILE_HEADER = 0;
: 858      1851 2 INVALIDATE (.HEADER);
: 859      1852 2 HEADER = READ_BLOCK (.CURRENT_VCB[VCBSL_IHDR2LBN], 1, HEADER_TYPE);
: 860      1853 2 IF NOT CHECK_HEADER2 (.HEADER, UPLIT WORD (FIDSC_INDEXF, FIDSC_INDEXF, 0))
: 861      1854 2 THEN
: 862      1855 2 BEGIN
: 863      1856 2 INVALIDATE (.HEADER);
: 864      1857 2 ERR_EXIT (0);
: 865      1858 2 END;
: 866      1859 2 IF FILE_SIZE (.HEADER) LSSU .FCB[FCBSL_FILESIZE]
: 867      1860 2 THEN ERR_EXIT (SS$ BADFILEHDR);
: 868      1861 2 FILE_HEADER = .HEADER;
: 869      1862 2 RESET_LBN (.HEADER, .FCB[FCBSL_HDLBN]);
: 870      1863 2 END;
: 871      1864 2
: 872      1865 2 USER_STATUS = .SAVE_STATUS;
: 873      1866 2
: 874      1867 2 .HEADER
: 875      1868 1 END;

```

! end of routine READ_IDX_HEADER

				00449	.BLKB	1		
		0000	0001	0001	0044A P.AAA:	.WORD	1, 1, 0	:
					.EXTRN	FILE_SIZE, READ_HEADER		
				000C 00000	.ENTRY	READ_IDX_HEADER, Save R2,R3		: 1788
CO	AA	80	AA	D0 00002	MOVL	-128(BASE), -64(BASE)		: 1843
	52	98	BA	D0 00007	MOVL	@-104(BASE), FCB		: 1845
			52	DD 0000B	PUSHL	FCB		: 1846
			7E	D4 0000D	CLRL	-(SP)		
0000G	CF		02	FB 0000F	CALLS	#2, READ_HEADER		
	53		50	D0 00014	MOVL	R0, HEADER		
			53	DD 00017	PUSHL	HEADER		: 1847
0000G	CF		01	FB 00019	CALLS	#1, FILE_SIZE		
	38	A2	50	D1 0001E	CML	R0, 56(FCB)		
			53	1E 00022	BGEQU	3\$		
		04	AA	D4 00024	CLRL	4(BASE)		: 1850
			53	DD 00027	PUSHL	HEADER		: 1851
0000G	CF		01	FB 00029	CALLS	#1, INVALIDATE		
	7E		01	7D 0002E	MOVQ	#1, -(SP)		: 1852
	50	98	AA	D0 00031	MOVL	-104(BASE), R0		
		2C	A0	DD 00035	PUSHL	44(R0)		
0000G	CF		03	FB 00038	CALLS	#3, READ_BLOCK		
	53		50	D0 0003D	MOVL	R0, HEADER		
		B7	AF	9F 00040	PUSHAB	P.AAA		: 1853
			53	DD 00043	PUSHL	HEADER		
0000G	CF		02	FB 00045	CALLS	#2, CHECK_HEADER2		
	0A		50	E8 0004A	BLBS	R0, 1\$		
			53	DD 0004D	PUSHL	HEADER		: 1856
0000G	CF		01	FB 0004F	CALLS	#1, INVALIDATE		


```

: 878 187C 1 GLOBAL ROUTINE MAP_IDX (VBN, COUNT) : 'L_NORM =
: 879 1871 1
: 880 1872 1 !++
: 881 1873 1
: 882 1874 1 FUNCTIONAL DESCRIPTION:
: 883 1875 1
: 884 1876 1 This routine maps a virtual block in the index file.
: 885 1877 1
: 886 1878 1 CALLING SEQUENCE:
: 887 1879 1 MAP_IDX (ARG1, ARG2)
: 888 1880 1
: 889 1881 1 INPUT PARAMETERS:
: 890 1882 1 ARG1: VBN of block to map
: 891 1883 1
: 892 1884 1 IMPLICIT INPUTS:
: 893 1885 1 NONE
: 894 1886 1
: 895 1887 1 OUTPUT PARAMETERS:
: 896 1888 1 COUNT: (optional) address to store count of contiguous blocks
: 897 1889 1
: 898 1890 1 IMPLICIT OUTPUTS:
: 899 1891 1 NONE
: 900 1892 1
: 901 1893 1 ROUTINE VALUE:
: 902 1894 1 LBN of blocks mapped or -1 if failure
: 903 1895 1
: 904 1896 1 SIDE EFFECTS:
: 905 1897 1 NONE
: 906 1898 1
: 907 1899 1 !--
: 908 1900 1
: 909 1901 2 BEGIN
: 910 1902 2
: 911 1903 2 EXTERNAL ROUTINE
: 912 1904 2 MAP_VBN : L_NORM, ! map VBN and turn window if necessary
: 913 1905 2 MAP_WINDOW : L_NORM, ! map VBN with current window
: 914 1906 2 RELEASE_SERIAL_LOCK : L_NORM, ! release sync lock on file
: 915 1907 2 SERIAL_FILE : L_NORM; ! get sync lock on file
: 916 1908 2
: 917 1909 2 LOCAL
: 918 1910 2 INCOMPLETE_FLAG, ! Saved state of CLF_INCOMPLETE
: 919 1911 2 IDX_FCB : REF BBLOCK, ! address of index file FCB
: 920 1912 2 LBN, ! resulting LBN from map
: 921 1913 2 UNMAPPED, ! received count of unmapped blocks
: 922 1914 2 TEMP; ! dummy to store resulting UCB
: 923 1915 2
: 924 1916 2 BIND_COMMON;
: 925 1917 2
: 926 1918 2 ! Try to map with the existing window first. This can be done without
: 927 1919 2 ! taking out the sync lock on the index file.
: 928 1920 2
: 929 1921 2
: 930 1922 2 IDX_FCB = .CURRENT_VCB [VCBSL_FCBFL];
: 931 1923 2
: 932 1924 2 IF (LBN = MAP_WINDOW (.VBN, .IDX_FCB [FCBSL_WLFL], 1000, UNMAPPED, TEMP))
: 933 1925 2 EQL -1
: 934 1926 2 THEN
```


0000G	CF	01	FB	00063	CALLS	#1, RELEASE SERIAL_LOCK
14	AA	6E	D0	00068	MOVL	TEMP, 20(BASE)
	02	6C	91	0006C	CMPB	(AP), #2
		0A	1F	0006F	BLSSU	2\$
08	BC	000003E8	8F	04	AE	C3 00071
			50		54	D0 0007B
					04	0007E

```

:
: 1939
: 1947
:
: 1948
: 1951
:

```

; Routine Size: 127 bytes, Routine Base: \$CODE\$ + 04D0

```

: 960      1952  1
: 961      1953  1 END
: 962      1954  0 ELUDOM

```

PSECT SUMMARY

Name	Bytes	Attributes
\$CODE\$	1359	NOVEC, NOWRT, RD, EXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2)

Library Statistics

File	Total	Symbols Loaded	Percent	Pages Mapped	Processing Time
_\$255\$DUA18:[SYSLIB]LIB.L32;1	18619	67	0	1000	00:02.0

COMMAND QUALIFIERS

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

```

: Size:      1352 code + 7 data bytes
: Run Time:   01:05.5
: Elapsed Time: 01:31.3
: Lines/CPU Min: 1791
: Lexemes/CPU-Min: 54177
: Memory Used: 337 pages
: Compilation Complete

```

