

[illegible]

```

LL          IIIIII          SSSSSSSS
LL          IIIIII          SSSSSSSS
LL          II             SS
LL          II             SS
LL          II             SS
LL          II             SS
LL          II             SSSSSS
LL          II             SSSSSS
LL          II             SS
LL          II             SS
LL          II             SS
LL          II             SS
LLLLLLLLLLLL IIIIII          SSSSSSSS
LLLLLLLLLLLL IIIIII          SSSSSSSS

```



```
1 0001 0 MODULE SHFDIR (
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 2
34 0034 1
35 0035 1 ABSTRACT:
36 0036 1
37 0037 1 This routine either extends or compresses a directory file.
38 0038 1
39 0039 1 ENVIRONMENT:
40 0040 1
41 0041 1 STARLET operating system, including privileged system services
42 0042 1 and internal exec routines.
43 0043 1
44 0044 1 --
45 0045 1
46 0046 1
47 0047 1 AUTHOR: Andrew C. Goldstein, CREATION DATE: 15-Apr-1977 13:25
48 0048 1
49 0049 1 MODIFIED BY:
50 0050 1
51 0051 1 V03-007 CDS0004 Christian D. Saether 23-Aug-1984
52 0052 1 Call routine to mark fcbs stale clusterwide when
53 0053 1 messing with the directory file.
54 0054 1
55 0055 1 V03-006 CDS0003 Christian D. Saether 11-Apr-1984
56 0056 1 When creating new buffers in shuffle operations, use
57 0057 1 the special lbn of -1 to avoid referencing real lbn's
```

```

: 58      0058 1  |
: 59      0059 1  |
: 60      0060 1  |
: 61      0061 1  |
: 62      0062 1  |
: 63      0063 1  |
: 64      0064 1  |
: 65      0065 1  |
: 66      0066 1  |
: 67      0067 1  |
: 68      0068 1  |
: 69      0069 1  |
: 70      0070 1  |
: 71      0071 1  |
: 72      0072 1  |
: 73      0073 1  |
: 74      0074 1  |
: 75      0075 1  |
: 76      0076 1  |
: 77      0077 1  |
: 78      0078 1  |
: 79      0079 1  |
: 80      0080 1  |
: 81      0081 1  |
: 82      0082 1  |
: 83      0083 1  |
: 84      0084 1  |
: 85      0085 1  |
: 86      0086 1  |
: 87      1077 1  |
: 88      1078 1  |
: 89      1079 1  |
: 90      1080 1  |
: 91      1081 1  |
: 92      1082 1  |

      in other directories.

V03-005 LMP0203      L. Mark Pilant,      29-Feb-1984 10:26
      Ignore the ACL segment in the header when re-initing the
      directory FCB resulting from a directory extension during
      a split operation.

V03-004 CDS0002      Christian D. Saether  30-Dec-1983
      Use L_NORM linkage and BIND_COMMON macro.

V03-003 ACG0367      Andrew C. Goldstein, 26-Oct-1983 19:51
      Update directory highwater mark when extending

V03-002 CDS0001      Christian D. Saether  3-Oct-1983
      Setup CURR_LCKINDX prior to reading directory header.

V03-001 ACG0271      Andrew C. Goldstein, 23-Mar-1982 9:57
      Return unique error on directory allocation failure

V02-007 ACG0208      Andrew C. Goldstein, 15-Oct-1981 18:31
      Add segmented directory record support

V02-006 ACG0167      Andrew C. Goldstein, 16-Apr-1980 19:26
      Previous revision history moved to F11B.REV

      **

      LIBRARY 'SYS$LIBRARY:LIB.L32';
      REQUIRE 'SRC$FCPDEF.B32';

      FORWARD ROUTINE
      SHUFFLE DIR      : L_NORM NOVALUE, ! extend or compress directory file
      FIX HEADER      : L_NORM NOVALUE, ! update directory file header & FCB
      HANDLER;         ! local condition handler
```



```

94 1083 1 GLOBAL ROUTINE SHUFFLE_DIR (DIRECTION) : L_NORM NOVALUE =
95 1084 1
96 1085 1 ++
97 1086 1
98 1087 1 FUNCTIONAL DESCRIPTION:
99 1088 1
100 1089 1     This routine extends or compresses a directory file. It is called
101 1090 1     to make free space in a block which is full or to squish out a block
102 1091 1     which is completely empty. If allocated but unused blocks are present
103 1092 1     at the end of the directory or if a squish is being done, the
104 1093 1     blocks are shuffled in place. Physically extending the directory
105 1094 1     requires reallocating and copying.
106 1095 1
107 1096 1 CALLING SEQUENCE:
108 1097 1     SHUFFLE_DIR (ARG1)
109 1098 1
110 1099 1 INPUT PARAMETERS:
111 1100 1     ARG1: +1 to extend the directory
112 1101 1     -1 to squish
113 1102 1
114 1103 1 IMPLICIT INPUTS:
115 1104 1     DIR_FCB: FCB of directory file
116 1105 1     DIR_VBN: current VBN of directory
117 1106 1     DIR_BUFFER: address of current directory block
118 1107 1     DIR_ENTRY: address of current directory record
119 1108 1     DIR_VERSION: address of version within record
120 1109 1
121 1110 1 OUTPUT PARAMETERS:
122 1111 1     NONE
123 1112 1
124 1113 1 IMPLICIT OUTPUTS:
125 1114 1     (for extension only)
126 1115 1     DIR_VBN: current VBN of directory, updated
127 1116 1     DIR_BUFFER: address of current directory block, updated
128 1117 1     DIR_ENTRY: address of current directory record, updated
129 1118 1     DIR_VERSION: address of version within record, updated
130 1119 1     DIR_END: address of end of data in directory block, updated
131 1120 1     (for compression)
132 1121 1     DIR_ENTRY: 0
133 1122 1
134 1123 1 ROUTINE VALUE:
135 1124 1     NONE
136 1125 1
137 1126 1 SIDE EFFECTS:
138 1127 1     directory extended, storage map altered, directory FCB altered
139 1128 1
140 1129 1 --
141 1130 1
142 1131 2 BEGIN
143 1132 2
144 1133 2 LINKAGE
145 1134 2     L_MAKE_POINTER = CALL :
146 1135 2     GLOBAL (MAP_POINTER = 9);
147 1136 2
148 1137 2 BUILTIN
149 1138 2     FP;
150 1139 2
```



```
151 1140 2 LABEL
152 1141 2 COPY_BLOCK; ! directory block copy code
153 1142 2
154 1143 2 GLOBAL REGISTER
155 1144 2 MAP_POINTER = 9 : REF BBLOCK; ! pointer to current retrieval pointer
156 1145 2
157 1146 2 LOCAL
158 1147 2 FIB : REF BBLOCK, ! address of FIB for this operation
159 1148 2 FCB : REF BBLOCK, ! address of FCB for directory
160 1149 2 HEADER : REF BBLOCK, ! address of directory file header
161 1150 2 IN_PLACE, ! in place copy flag
162 1151 2 NEW_SIZE, ! size to extend directory to
163 1152 2 NEW_LBN, ! starting LBN of new space
164 1153 2 BUFFER : REF BBLOCK, ! buffer address of current directory block
165 1154 2 COMP_BUFFER, ! buffer address used for compression
166 1155 2 NEW_BUFFER, ! address of newly read expanded block
167 1156 2 OFFSET, ! VBN offset in shuffle copy
168 1157 2 P1 : REF BBLOCK, ! directory record pointer
169 1158 2 END_POINT : REF BBLOCK, ! pointer to end of data in block
170 1159 2 REC_OFFSET, ! block offset of current record
171 1160 2 VER_OFFSET; ! block offset of current version
172 1161 2
173 1162 2 BIND_COMMON;
174 1163 2
175 1164 2 DIR_CONTEXT_DEF;
176 1165 2
177 1166 2 EXTERNAL ROUTINE
178 1167 2 MAKE_FCB_STALE : L_NORM NOVALUE, ! mark fcb as stale clusterwide.
179 1168 2 SAVE_CONTEXT : L_NORM, ! save reentrant context area
180 1169 2 RESTORE_CONTEXT : L_NORM, ! restore reentrant context area
181 1170 2 READ_HEADER : L_NORM, ! read file header
182 1171 2 CHARGE_QUOTA : L_NORM, ! charge space to user's quota
183 1172 2 ALLOC_BLOCKS : L_NORM, ! allocate blocks from storage map
184 1173 2 MAKE_POINTER : L_MAKE_POINTER, ! build header map pointer
185 1174 2 READ_BLOCK : L_NORM, ! read a disk block
186 1175 2 RESET_LBN : L_NORM, ! assign new LBN to buffer
187 1176 2 WRITE_BLOCK : L_NORM, ! write block to disk
188 1177 2 CREATE_BLOCK : L_NORM, ! fabricate a block buffer
189 1178 2 INVALIDATE : L_NORM, ! invalidate a block buffer
190 1179 2 NEXT_REC : L_NORM, ! get next directory record
191 1180 2 TRUNCATE_HEADER : L_NORM, ! truncate file header
192 1181 2 ZERO_WINDOWS : L_NORM, ! invalidate related file windows
193 1182 2
194 1183 2
195 1184 2 ! First save the current context, since this is a secondary file operation.
196 1185 2 ! Set up the secondary context pointers. Then read the directory file header.
197 1186 2
198 1187 2
199 1188 2 SAVE_CONTEXT ();
200 1189 2 PRIMARY_FCB = FCB = .DIR_FCB;
201 1190 2
202 1191 2 MAKE_FCB_STALE (.FCB);
203 1192 2
204 1193 2 CURR_LCKINDX = .DIR_LCKINDX;
205 1194 2 FIB = SECOND FIB;
206 1195 2 CH$MOVE (FIB$$_FID, LOCAL_FIB[FIB$W_DID], FIB[FIB$W_FID]);
207 1196 2
```



```
208 1197 2 HEADER = READ_HEADER (0, .FCB);
209 1198 2
210 1199 2 ! If the directory is being expanded, see if space is available. If not,
211 1200 2 ! allocate a new larger area. The next VBN to use is the current directory
212 1201 2 ! EOF block number. If the block is not present in the file, the directory
213 1202 2 ! must be physically extended.
214 1203 2
215 1204 2
216 1205 3 COPY_BLOCK: BEGIN
217 1206 3 IN_PLACE = 1; ! assume in place shuffle
218 1207 3 IF .DIRECTION GTR 0
219 1208 3 THEN
220 1209 4 BEGIN
221 1210 4 IF .FCB[FCB$L_EFBLK] + 1 GTRU .FCB[FCB$L_FILESIZE]
222 1211 4 THEN
223 1212 5 BEGIN
224 1213 5
225 1214 5 ! Compute the number of blocks needed (50% of the current directory size)
226 1215 5 ! and allocate the new space contiguously. Limit the number of blocks
227 1216 5 ! allocated to what will fit in the map area of the header.
228 1217 5
229 1218 5
230 1219 5 IN_PLACE = 0;
231 1220 5 NEW_SIZE = .FCB[FCB$L_FILESIZE] + MAXU (.FCB[FCB$L_FILESIZE]/2, 1);
232 1221 5 IF .FCB[FCB$L_FILESIZE] GEQU 1024 THEN ERR_EXIT (SS$DIRFULL);
233 1222 5 IF .NEW_SIZE GTRU 1024 THEN NEW_SIZE = 1024;
234 1223 5 CHARGE_QUOTA (.HEADER[FH2$L_FILEOWNER], .NEW_SIZE - .FCB[FCB$L_FILESIZE],
235 1224 5 BITLIST (QUOTA_CHECK));
236 1225 5
237 1226 5 FIB[FIB$V_ALCON] = 1;
238 1227 5 FIB[FIB$V_FILCON] = 1;
239 1228 5 IF NOT ALLOC_BLOCKS (.FIB, .NEW_SIZE, NEW_LBN, NEW_SIZE)
240 1229 5 THEN ERR_EXIT (SS$DIRALLOC);
241 1230 5 UNREC_COUNT = .NEW_SIZE;
242 1231 5 UNREC_LBN = .NEW_LBN;
243 1232 5 UNREC_RVN = .CURRENT_RVN;
244 1233 5 END
245 1234 4 ELSE
246 1235 4 NEW_LBN = .FCB[FCB$L_STLBN]; ! use existing space
247 1236 4
248 1237 4 ! Now copy the directory blocks (following code is for expansion). For
249 1238 4 ! maximum safety, we copy in reverse order, so that if the operation fails
250 1239 4 ! or the system crashes, we have duplicate, rather than missing, directory
251 1240 4 ! entries. We update the file header after the first block has been
252 1241 4 ! written in an in place copy for the same reason.
253 1242 4
254 1243 4
255 1244 4 OFFSET = 0; ! shuffle offset
256 1245 4 DECR VBN FROM .FCB[FCB$L_EFBLK] TO 1 DO
257 1246 5 BEGIN
258 1247 5 BUFFER = READ_BLOCK (.VBN+.FCB[FCB$L_STLBN]-1, 1, DIRECTORY_TYPE);
259 1248 5
260 1249 5 ! When we reach the VBN at which the shuffler was called, we split the
261 1250 5 ! block.
262 1251 5 ! We find the first record past the midpoint (past the 3/4 mark if
263 1252 5 ! this is the last block) and split into two buffers. Note that this logic
264 1253 5 ! depends on the availability from the buffer manager two blocks of
```



```
265 1254 5 ! adjacent LBN's referenced in direct sequence.
266 1255 5 !
267 1256 5
268 1257 5
269 1258 5
270 1259 6
271 1260 6
272 1261 6
273 1262 6
274 1263 6
275 1264 6
276 1265 6
277 1266 6
278 1267 6
279 1268 6
280 1269 6
281 1270 7
282 1271 7
283 1272 7
284 1273 6
285 1274 6
286 1275 6 !
287 1276 6 !
288 1277 6 !
289 1278 6 !
290 1279 6 !
291 1280 6 !
292 1281 6 !
293 1282 6 !
294 1283 6 !
295 1284 6 !
296 1285 6 !
297 1286 6 !
298 1287 6 !
299 1288 6 !
300 1289 6 !
301 1290 6 !
302 1291 6 !
303 1292 6 !
304 1293 6 !
305 1294 6 !
306 1295 6 !
307 1296 7 !
308 1297 7 !
309 1298 7 !
310 1299 7 !
311 1300 7 !
312 1301 7 !
313 1302 7 !
314 1303 6 !
315 1304 6 !
316 1305 6 !
317 1306 7 !
318 1307 7 !
319 1308 7 !
320 1309 8 !
321 1310 8 !
```

```
IF .VBN EQL .DIR_VBN
THEN
  BEGIN
    LOCAL P2, BUFFER2 : REF BBLOCK, LIMIT, HEADER_LENGTH;
    REC_OFFSET = .DIR_ENTRY - .DIR_BUFFER;
    VER_OFFSET = .DIR_VERSION - .DIR_BUFFER;

    P1 = .BUFFER;
    LIMIT = .BUFFER + 256;
    IF .DIR_VBN EQL .FCB[FCB$L_EFBLK]
    THEN LIMIT = .LIMIT + 128;

    UNTIL .P1 GEQA .LIMIT DO
      BEGIN
        P2 = .P1;
        P1 = NEXT_REC (.P1);
      END;

    ! The point at which we split depends on the the relation between the
    ! location of the new entry and the two split pointers.

    !-----+-----+-----+
    ! |               | record |               |
    !-----+-----+-----+
    !               P2       P1

    P1 is now at the record boundary just past the half (or 3/4) mark,
    and P2 is at the previous record boundary. If the new entry precedes
    P2, we split at P2 and the entry goes into the former block. If the
    new entry is at P1 or later, we split at P1 and the entry goes into
    the latter block. If the entry is a new version of the record at P2,
    we split at P1 unless the record is already the last in its block, in
    which case we split at P2. Finally, if we discover that P1 is at the
    end of the block and P2 is at the beginning (i.e., the block contains
    one large record), we have to split the record instead. Got it?

    IF .REC_OFFSET + .BUFFER GTRA .P2
    THEN
      BEGIN
        P2 = .P1;
        REC_OFFSET = .REC_OFFSET + .BUFFER - .P2;
        VER_OFFSET = .VER_OFFSET + .BUFFER - .P2;
        DIR_VBN = .DIR_VBN + 1;
      END

    ELSE IF .REC_OFFSET + .BUFFER EQL .P2
    AND .DIR_VERSION NEQ 0
    THEN
      BEGIN
        IF .P1[DIR$W_SIZE] EQL 65535
        THEN
          BEGIN
            IF .P2 EQL .BUFFER
```



```

322      1311 8      THEN
323      1312 8      P2 = 0      ! flag need for record split
324      1313 8      ELSE
325      1314 9      BEGIN
326      1315 9      REC_OFFSET = .REC_OFFSET + .BUFFER - .P2;
327      1316 9      VER_OFFSET = .VER_OFFSET + .BUFFER - .P2;
328      1317 9      DIR_VBN = .DIR_VBN + 1;
329      1318 8      END;
330      1319 8      END
331      1320 7      ELSE
332      1321 7      P2 = .P1;
333      1322 6      END;
334      1323 6
335      1324 6 ! We now do either a block or record split; the latter is flagged here
336      1325 6 ! by P2 set to zero. The block split is performed by allocating a new
337      1326 6 ! buffer and copying the records past the split point (now denoted by P2)
338      1327 6 ! into the new buffer, and then erasing the copied records from the
339      1328 6 ! first buffer.
340      1329 6
341      1330 6
342      1331 6      IF .P2 NEQ 0
343      1332 6      THEN
344      1333 7      BEGIN
345      1334 7      P1 = .P2 - .BUFFER;      ! save split offset point
346      1335 7
347      1336 7 ! Use the special lbn of -1 for the new buffer. A reset_lbn call will
348      1337 7 ! establish the correct lbn prior to writing it.
349      1338 7
350      1339 7
351      1340 7      BUFFER2 = CREATE_BLOCK (-1, 1, DIRECTORY_TYPE);
352      1341 7      CH$MOVE (512-.P1, .P2, .BUFFER2);
353      1342 7      CH$COPY (2, UPLIT WORD (-1), 0, 512-.P1, .P2);
354      1343 7      END
355      1344 7
356      1345 7 ! Otherwise we do a record split. The principle is similar to the block
357      1346 7 ! split, except that we always split at the point of version insertion.
358      1347 7 ! Assumptions: the current directory buffer contains only one record,
359      1348 7 ! which fills it, and we are adding a new version. Allocate a buffer and
360      1349 7 ! duplicate the block in it to set up the record header. Then carve out
361      1350 7 ! the appropriate set of versions from each record and flag the records.
362      1351 7
363      1352 7
364      1353 6      ELSE
365      1354 7      BEGIN
366      1355 7      P1 = .P1 - .BUFFER;
367      1356 7      HEADER_LENGTH = (.BUFFER[DIR$B_NAMECOUNT] + 1 AND NOT 1) + DIR$C_LENGTH;
368      1357 7
369      1358 7 ! Use the special lbn of -1 for the new buffer. A reset_lbn call will
370      1359 7 ! establish the correct lbn prior to writing it.
371      1360 7
372      1361 7
373      1362 7      BUFFER2 = CREATE_BLOCK (-1, 1, DIRECTORY_TYPE);
374      1363 7      CH$MOVE (512, .BUFFER, .BUFFER2);
375      1364 7      CH$COPY (2, UPLIT WORD (-1), 0,
376      1365 7      512-.VER_OFFSET, .BUFFER+.VER_OFFSET);
377      1366 7      CH$COPY (.P1-.VER_OFFSET+2, .BUFFER2+.VER_OFFSET, 0,
378      1367 7      512-HEADER_LENGTH, .BUFFER2+HEADER_LENGTH);
```



```
379 1368 7
380 1369 7      BUFFER[DIR$W_SIZE] = .VER_OFFSET - 2;
381 1370 7      BUFFER2[DIR$W_SIZE] = .P1 - .VER_OFFSET + .HEADER_LENGTH - 2;
382 1371 7
383 1372 7      Pick the record to contain the new entry. We use the smaller of the
384 1373 7      two.
385 1374 7
386 1375 7
387 1376 7      IF .VER_OFFSET - .HEADER_LENGTH GTRU .P1 - .VER_OFFSET
388 1377 7      THEN
389 1378 8          BEGIN
390 1379 8              DIR_VBN = .DIR_VBN + 1;
391 1380 8              VER_OFFSET = .HEADER_LENGTH;
392 1381 7          END;
393 1382 6      END;
394 1383 6
395 1384 6      Set the higher block to its new LBN and write it. Bump the offset so
396 1385 6      that subsequent file to file copies now go to corresponding VBN's.
397 1386 6      If this is an in place copy then fix up the header now.
398 1387 6
399 1388 6
400 1389 6      RESET_LBN (.BUFFER2, .VBN+.NEW_LBN);
401 1390 6      OFFSET = -1;
402 1391 6      WRITE_BLOCK (.BUFFER2);
403 1392 6      IF .IN_PLACE AND .VBN EQL .FCB[FCB$L_EFBLK]
404 1393 6      THEN FIX_HEADER (.HEADER, 1);
405 1394 5      END;
406 1395 5      ! end of block split condition
407 1396 5
408 1397 5      For the rest of the copy loop, assign the output LBN to the buffer and
409 1398 5      write it. On an in place copy, kick out of the loop after we have done
410 1399 5      the block split. Otherwise, fix the header after the first write.
411 1400 5
412 1401 5      RESET_LBN (.BUFFER, .VBN+.NEW_LBN+.OFFSET);
413 1402 5      WRITE_BLOCK (.BUFFER);
414 1403 5      IF .IN_PLACE
415 1404 5      THEN
416 1405 6          BEGIN
417 1406 6              IF .OFFSET NEQ 0
418 1407 6              THEN LEAVE COPY_BLOCK
419 1408 6              ELSE IF .VBN EQL .FCB[FCB$L_EFBLK]
420 1409 6              THEN FIX_HEADER (.HEADER, 1);
421 1410 5          END;
422 1411 4      END;
423 1412 4      ! of in place copy loop
424 1413 4
425 1414 4      Now deallocate the old directory blocks. Then build retrieval pointers
426 1415 4      for the new blocks in the file header. Do the truncation with a local
427 1416 4      condition handler enabled for special error recovery.
428 1417 4
429 1418 4      .FP = HANDLER;
430 1419 4      TRUNCATE_HEADER (.FIB, .HEADER);
431 1420 4      .FP = 0;
432 1421 4
433 1422 4      HEADER[FH2$B_MAP_INUSE] = 0;
434 1423 4      CH$FILL (0, 7.HEADER[FH2$B_ACOFFSET] - .HEADER[FH2$B_MPOFFSET]) * 2,
435 1424 4      .HEADER + .HEADER[FH2$B_MPOFFSET]*2);
```



```
436 1425 4 MAP_POINTER = .HEADER + .HEADER[FH2$B_MPOFFSET]*2;
437 1426 4 MAKE_POINTER (.NEW_SIZE, .NEW_LBN, .HEADER);
438 1427 4
439 1428 4 UNREC_COUNT = 0;
440 1429 4 KERNEL_CALL (ZERO WINDOWS, .FCB);
441 1430 4 CHARGE_QUOTA (.HEADER[FH2$L_FILEOWNER], .NEW_SIZE - .FCB[FCB$L_FILESIZE],
442 1431 4 BITLIST (QUOTA_CHARGE));
443 1432 4
444 1433 4 END ! of expansion conditional
445 1434 4
446 1435 4 ! For an inplace compression, we copy the blocks in forward order, again
447 1436 4 ! for maximum safety.
448 1437 4
449 1438 4
450 1439 3 ELSE
451 1440 4 BEGIN
452 1441 4 DIR_ENTRY = 0; ! indicate no block resident
453 1442 4 INVALIDATE (.DIR_BUFFER); ! punt the block being squished out
454 1443 4 INCR_VBN FROM .DIR_VBN+1 TO .FCB[FCB$L_EFBLK] DO
455 1444 5 BEGIN
456 1445 5 COMP_BUFFER = READ_BLOCK (.VBN+.FCB[FCB$L_STLBN]-1, 1, DIRECTORY_TYPE);
457 1446 5 RESET_LBN (.COMP_BUFFER, .VBN+.FCB[FCB$L_STLBN]-2);
458 1447 5 WRITE_BLOCK (.COMP_BUFFER);
459 1448 5 END;
460 1449 3 END;
461 1450 3
462 1451 3 ! Now, for an extension copy or a compression, update the file header.
463 1452 3
464 1453 3
465 1454 3 FIX_HEADER (.HEADER, .DIRECTION);
466 1455 3
467 1456 2 END; ! end of block COPY_BLOCK
468 1457 2
469 1458 2 ! For an extension, read back the block where the new entry will go
470 1459 2 ! and update the lookup pointers. For a compression, save the pointer
471 1460 2 ! in offset form so it can be used by the remove cleanup in case of error.
472 1461 2
473 1462 2
474 1463 2 IF .DIRECTION GTR 0
475 1464 2 THEN
476 1465 3 BEGIN
477 1466 3 NEW_BUFFER = READ_BLOCK (.DIR_VBN+.FCB[FCB$L_STLBN]-1, 1, DIRECTORY_TYPE);
478 1467 3 DIR_BUFFER = .NEW_BUFFER;
479 1468 3 DIR_ENTRY = .REC_OFFSET + .NEW_BUFFER;
480 1469 3 IF .DIR_VERSION NEQ 0
481 1470 3 THEN DIR_VERSION = .NEW_BUFFER + .VER_OFFSET;
482 1471 3
483 1472 3 END_POINT = .NEW_BUFFER;
484 1473 3 UNTIL .END_POINT[DIR$W_SIZE] EQL 65535 DO
485 1474 3 END_POINT = NEXT_REC (.END_POINT);
486 1475 3 DIR_END = .END_POINT + 2;
487 1476 3 END;
488 1477 2
489 1478 2 ! Finally switch back to primary context.
490 1479 2
491 1480 2
492 1481 2 RESTORE_CONTEXT ();
```


: 493
: 494

1482 2
1483 1 END;

N 2
16-Sep-1984 01:10:06
14-Sep-1984 12:30:47

VAX-11 Bliss-32 V4.0-742 Page 10
DISK\$VMSMASTER:[F11X.SRC]SHFDIR.B32;1 (2)

```
! end of routine EXTEND_DIR
```

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

```
FFFF 00000 P.AAA: .WORD -1
FFFF 00002 P.AAB: .WORD -1
```

```
.EXTRN MAKE_FCB_STALE, SAVE_CONTEXT
.EXTRN RESTORE_CONTEXT
.EXTRN READ_HEADER, CHARGE_QUOTA
.EXTRN ALLOC_BLOCKS, MAKE_POINTER
.EXTRN READ_BLOCK, RESET_CBN
.EXTRN WRITE_BLOCK, CREATE_BLOCK
.EXTRN INVALIDATE, NEXT_REC
.EXTRN TRUNCATE_HEADER
.EXTRN ZERO_WINDOWS
```

OBFC 00000

.ENTRY	SHUFFLE_DIR, Save R2,R3,R4,R5,R6,R7,R8,R9,-	1083
	R11	
SUBL2	#40, SP	
MOVAB	220(BASE), 12(SP)	1160
MOVAB	516(BASE), R2	
CALLS	#0, SAVE_CONTEXT	1186
MOVL	208(BASE), FCB	1189
MOVL	FCB, 8(BASE)	
PUSHL	FCB	1191
CALLS	#1, MAKE_FCB_STALE	
MOVL	212(BASE), 20(BASE)	1193
MOVAB	580(BASE), FIB	1194
ADDL3	#4, FIB, R7	1195
MOVC3	#6, 10(R2), (R7)	
PUSHL	FCB	1197
CLRL	-(SP)	
CALLS	#2, READ_HEADER	
MOVL	R0, HEADER	
MOVL	#1, IN_PLACE	1206
TSTL	DIRECTION	1207
BGTR	1\$	
BRW	24\$	
ADDL3	#1, 60(FCB), R0	1210
CMPL	R0, 56(FCB)	
BLEQU	6\$	
CLRL	IN_PLACE	1219
DIVL3	#2, 56(FCB), R0	1220
BNEQ	2\$	
MOVL	#1, R0	
MOVAB	@56(FCB)[R0], NEW_SIZE	
CMPL	56(FCB), #1024	1221
BLSSU	3\$	
CHMU	#2144	
RET		
CMPL	NEW_SIZE, #1024	1222

	20	AE	0400	06	1B	00087	BLEQU	4\$		
				8F	3C	00089	MOVZWL	#1024, NEW_SIZE		
				01	DD	0008F	PUSHL	#1		1224
7E	24	AE	38	A6	C3	00091	SUBL3	56(FCB), NEW_SIZE, -(SP)		1223
52	0C	AE		3C	C1	00097	ADDL3	#60, HEADER, -R2		
				62	DD	0009C	PUSHL	(R2)		
	0000G	CF		03	FB	0009E	CALLS	#3, CHARGE_QUOTA		
50	08	AE		16	C1	000A3	ADDL3	#22, FIB, R0		1226
		60		01	88	000A8	BISB2	#1, (R0)		
50	08	AE		16	C1	000AB	ADDL3	#22, FIB, R0		1227
		60		04	88	000B0	BISB2	#4, (R0)		
			20	AE	9F	000B3	PUSHAB	NEW_SIZE		1228
			28	AE	9F	000B6	PUSHAB	NEW_LBN		
			28	AE	DD	000B9	PUSHL	NEW_SIZE		
			14	AE	DD	000BC	PUSHL	FIB		
	0000G	CF		04	FB	000BF	CALLS	#4, ALLOC_BLOCKS		
		05		50	E8	000C4	BLBS	R0, 5\$		
			09C8	8F	BF	000C7	CHMU	#2504		1229
					04	000CB	RET			
	28	AA	20	AE	D0	000CC	MOVL	NEW_SIZE, 40(BASE)		1230
	24	AA	24	AE	D0	000D1	MOVL	NEW_LBN, 36(BASE)		1231
	2C	AA	A0	AA	D0	000D6	MOVL	-96(BASE), 44(BASE)		1232
				05	11	000DB	BRB	7\$		1210
	24	AE	30	A6	D0	000DD	MOVL	48(FCB), NEW_LBN		1235
			1C	AE	D4	000E2	CLRL	OFFSET		1244
57	3C	A6		01	C1	000E5	ADDL3	#1, 60(FCB), VBN		1245
				01DE	31	000EA	BRW	21\$		
				02	DD	000ED	PUSHL	#2		1247
				01	DD	000EF	PUSHL	#1		
50		57	30	A6	C1	000F1	ADDL3	48(FCB), VBN, R0		
			FF	A0	9F	000F6	PUSHAB	-1(R0)		
	0000G	CF		03	FB	000F9	CALLS	#3, READ_BLOCK		
	14	AE		50	D0	000FE	MOVL	R0, BUFFER		
	0C	BE		57	D1	00102	CMPL	VBN, @12(SP)		1257
				03	13	00106	BEQL	9\$		
				018C	31	00108	BRW	19\$		
50	0C	AE		08	C1	0010B	ADDL3	#8, 12(SP), R0		1261
51	0C	AE		04	C1	00110	ADDL3	#4, 12(SP), R1		
5B		60		61	C3	00115	SUBL3	(R1), (R0), REC_OFFSET		
50	0C	AE		0C	C1	00119	ADDL3	#12, 12(SP), R0		1262
51	0C	AE		04	C1	0011E	ADDL3	#4, 12(SP), R1		
58		60		61	C3	00123	SUBL3	(R1), (R0), VER_OFFSET		
	18	AE	14	AE	D0	00127	MOVL	BUFFER, P1		1264
52	14	AE	00000100	8F	C1	0012C	ADDL3	#256, BUFFER, LIMIT		1265
	3C	A6	0C	BE	D1	00135	CMPL	@12(SP), 60(FCB)		1266
				05	12	0013A	BNEQ	10\$		
			0080	C2	9E	0013C	MOVAB	128(R2), LIMIT		1267
			18	AE	D1	00141	CMPL	P1, LIMIT		1269
				12	1E	00145	BGEQU	11\$		
				18	AE	D0	00147	MOVL	P1, P2	1271
				18	AE	DD	0014B	PUSHL	P1	1272
	0000G	CF		01	FB	0014E	CALLS	#1, NEXT_REC		
	18	AE		50	D0	00153	MOVL	R0, P1		
				E8	11	00157	BRB	10\$		1269
50		5B	14	AE	C1	00159	ADDL3	BUFFER, REC_OFFSET, R0		1294
		59		50	D1	0015E	CMPL	R0, P2		
				13	1B	00161	BLEQU	12\$		

5B	59	18	AE	D0	00163	MOVL	P1, P2	1297	
51	50		59	C3	00167	SUBL3	P2, R0, REC_OFFSET	1298	
58	58	14	AE	C1	0016B	ADDL3	BUFFER, VER_OFFSET, R1	1299	
	51		59	C3	00170	SUBL3	P2, R1, VER_OFFSET		
			2A	11	00174	BRB	14\$	1300	
			31	12	00176	BNEQ	16\$	1303	
51	OC	AE	OC	C1	00178	ADDL3	#12, 12(SP), R1	1304	
			61	D5	0017D	TSTL	(R1)		
	FFFF	8F	18	BE	B1	00181	BEQL	16\$	
			1C	12	00187	CMPW	@P1, #65535	1307	
	14	AE	59	D1	00189	BNEQ	15\$		
			04	12	0018D	CMPL	P2, BUFFER	1310	
			59	D4	0018F	BNEQ	13\$		
			16	11	00191	CLRL	P2	1312	
5B		50	59	C3	00193	BRB	16\$		
50		58	14	AE	C1	00197	SUBL3	P2, R0, REC_OFFSET	1315
58		50	59	C3	0019C	ADDL3	BUFFER, VER_OFFSET, R0	1316	
			OC	BE	D6	001A0	SUBL3	P2, R0, VER_OFFSET	
		59	18	AE	D0	001A5	INCL	@12(SP)	1317
			59	D5	001A9	BRB	16\$	1307	
			38	13	001AB	MOVL	P1, P2	1321	
18	AE	59	14	AE	C3	001AD	TSTL	P2	1331
				02	DD	001B3	BEQL	17\$	
				01	DD	001B5	SUBL3	BUFFER, P2, P1	1334
				01	CE	001B7	PUSHL	#2	1340
	0000G	7E		03	FB	001BA	PUSHL	#1	
	10	AE		50	D0	001BF	MNEGL	#1, -(SP)	
10	50	00000200	18	AE	C3	001C3	CALLS	#3, CREATE_BLOCK	
	BE	69	18	50	28	001CC	MOVL	R0, BUFFER2	1341
	50	00000200	18	AE	C3	001D1	SUBL3	P1, #512, R0	
50	00	FE1D		02	2C	001DA	MOVC3	R0, (P2), @BUFFER2	1342
				69		001E1	SUBL3	P1, #512, R0	
				0087	31	001E2	MOVC5	#2, P.AAA, #0, R0, (P2)	
		18	14	AE	C2	001E5	BRW	18\$	1331
		14		05	C1	001EA	SUBL2	BUFFER, P1	1355
		51		60	9A	001EF	ADDL3	#5, BUFFER, R0	1356
				51	D6	001F2	MOVZBL	(R0), R1	
		51		01	8A	001F4	INCL	R1	
		59	06	A1	9E	001F7	BICB2	#1, R1	
				02	DD	001FB	MOVAB	6(R1), HEADER_LENGTH	
				01	DD	001FD	PUSHL	#2	1362
				01	CE	001FF	PUSHL	#1	
	0000G	7E		03	FB	00202	MNEGL	#1, -(SP)	
	10	AE		50	D0	00207	CALLS	#3, CREATE_BLOCK	
10	BE	14	0200	8F	28	0020B	MOVL	R0, BUFFER2	
	50	00000200		58	C3	00213	MOVC3	#512, @BUFFER, @BUFFER2	1363
			14	BE48	9F	0021B	SUBL3	VER_OFFSET, #512, R0	1365
	00	FDDA		02	2C	0021F	PUSHAB	@BUFFER[R8]	
				9E		00226	MOVC5	#2, P.AAB, #0, R0, @ (SP)+	
	50	18		58	C3	00227			
	51	00000200		02	C0	0022C	SUBL3	VER_OFFSET, P1, R0	1366
				59	C3	0022F	ADDL2	#2, R0	
			10	BE49	9F	00237	SUBL3	HEADER_LENGTH, #512, R1	1367
			14	BE48	9F	0023B	PUSHAB	@BUFFER2[R9]	
51	00	9E		50	2C	0023F	PUSHAB	@BUFFER2[R8]	
							MOVC5	R0, @ (SP)+, #0, R1, @ (SP)+	

51	04	AE	01	C1	0030D	ADDL3	#1, HEADER, R1	1425
		50	61	9A	00312	MOVZBL	(R1), R0	
		59	04	BE	40	3E	00315	
			04	AE	DD	0031A	MOVAV	@HEADER[R0], MAP_POINTER
			28	AE	DD	0031D	PUSHL	HEADER
			28	AE	DD	00320	PUSHL	NEW_LBN
	0000G	CF	03	FB	00323	PUSHL	NEW_SIZE	
			28	AA	D4	00328	CALLS	#3, MAKE_POINTER
				56	DD	0032B	CLRL	40(BASE)
	0000G	CF	01	FB	0032D	PUSHL	FCB	
			02	DD	00332	CALLS	#1, ZERO_WINDOWS	
7E	24	AE	38	A6	C3	00334	PUSHL	#2
52	0C	AE		3C	C1	0033A	SUBL3	56(FCB), NEW_SIZE, -(SP)
				62	DD	0033F	ADDL3	#60, HEADER, R2
	0000G	CF	03	FB	00341	PUSHL	(R2)	
			4B	11	00346	CALLS	#3, CHARGE_QUOTA	
50	0C	AE	08	C1	00348	BRB	27\$	1207
			60	D4	0034D	ADDL3	#8, 12(SP), R0	1441
52	0C	AE	04	C1	0034F	CLRL	(R0)	
			62	DD	00354	ADDL3	#4, 12(SP), R2	1442
	0000G	CF	01	FB	00356	PUSHL	(R2)	
			3C	A6	D0	0035B	CALLS	#1, INVALIDATE
	53		0C	BE	D0	0035F	MOVL	60(FCB), R3
	52			2A	11	00363	MOVL	@12(SP), VBN
				02	DD	00365	BRB	26\$
				01	DD	00367	PUSHL	#2
50		52	30	A6	C1	00369	PUSHL	#1
			FF	A0	9F	0036E	ADDL3	48(FCB), VBN, R0
	0000G	CF		03	FB	00371	PUSHAB	-1(R0)
				50	D0	00376	CALLS	#3, READ_BLOCK
50		52	30	A6	C1	00379	MOVL	R0, COMP_BUFFER
			FE	A0	9F	0037E	ADDL3	48(FCB), VBN, R0
				54	DD	00381	PUSHAB	-2(R0)
	0000G	CF		02	FB	00383	PUSHL	COMP_BUFFER
				54	DD	00388	CALLS	#2, RESET_LBN
	0000G	CF		01	FB	0038A	PUSHL	COMP_BUFFER
D2		52		53	F3	0038F	CALLS	#1, WRITE_BLOCK
			04	AC	DD	00393	AOBLEQ	R3, VBN, 25\$
			08	AE	DD	00396	PUSHL	DIRECTION
				02	FB	00399	PUSHL	HEADER
	0000V	CF	04	AC	D5	0039E	CALLS	#2, FIX_HEADER
				4E	15	003A1	TSTL	DIRECTION
				02	DD	003A3	BLEQ	31\$
				01	DD	003A5	PUSHL	#2
50	14	BE	30	A6	C1	003A7	PUSHL	#1
			FF	A0	9F	003AD	ADDL3	48(FCB), @20(SP), R0
	0000G	CF		03	FB	003B0	PUSHAB	-1(R0)
51	0C	AE		04	C1	003B5	CALLS	#3, READ_BLOCK
				50	D0	003BA	ADDL3	#4, 12(SP), R1
51	0C	AE		08	C1	003BD	MOVL	NEW_BUFFER, (R1)
61		5B		50	C1	003C2	ADDL3	#8, 12(SP), R1
51	0C	AE		0C	C1	003C6	ADDL3	NEW_BUFFER, REC_OFFSET, (R1)
				61	D5	003CB	ADDL3	#12, 12(SP), R1
				09	13	003CD	TSTL	(R1)
51	0C	AE		0C	C1	003CF	BEQL	29\$
61		50		58	C1	003D4	ADDL3	#12, 12(SP), R1
	FFFF	8F		60	B1	003D8	ADDL3	VER_OFFSET, NEW_BUFFER, (R1)
						29\$:	CMPL	(END_POINT), #65535

SHFDIR
V04-000

F 3
16-Sep-1984 01:10:06
14-Sep-1984 12:30:47

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[F11X.SRC]SHFDIR.B32;1 Page 15
(2)

			09	13	003DD	BEQL	30\$	:	
			50	DD	003DF	PUSHL	END_POINT	:	1474
	0000G	CF	01	FB	003E1	CALLS	#1-NEXT_REC	:	
			F0	11	003E6	BRB	29\$	:	
51	0C	AE	10	C1	003E8	ADDL3	#16, 12(SP), R1	:	1475
		61	A0	9E	003ED	MOVAB	2(R0), (R1)	:	
	0000G	CF	00	FB	003F1	CALLS	#0, RESTORE_CONTEXT	:	1481
				04	003F6	RET		:	1483

; Routine Size: 1015 bytes, Routine Base: \$CODE\$ + 0004


```

: 496 1484 1 ROUTINE FIX_HEADER (HEADER, DIRECTION) : L_NORM NOVALUE =
: 497 1485 1
: 498 1486 1 ++
: 499 1487 1
: 500 1488 1 FUNCTIONAL DESCRIPTION:
: 501 1489 1
: 502 1490 1 This routine updates the file header of the directory being
: 503 1491 1 extended or compressed, and also updates the directory FCB.
: 504 1492 1
: 505 1493 1
: 506 1494 1 CALLING SEQUENCE:
: 507 1495 1 FIX_HEADER (ARG1, ARG2)
: 508 1496 1
: 509 1497 1 INPUT PARAMETERS:
: 510 1498 1 ARG1: address of file header
: 511 1499 1 ARG2: +1 for extension
: 512 1500 1 -1 for compression
: 513 1501 1
: 514 1502 1 IMPLICIT INPUTS:
: 515 1503 1 DIR_VBN: VBN of directory being split or compressed out
: 516 1504 1 DIR_FCB: address of directory FCB
: 517 1505 1
: 518 1506 1 OUTPUT PARAMETERS:
: 519 1507 1 NONE
: 520 1508 1
: 521 1509 1 IMPLICIT OUTPUTS:
: 522 1510 1 NONE
: 523 1511 1
: 524 1512 1 ROUTINE VALUE:
: 525 1513 1 NONE
: 526 1514 1
: 527 1515 1 SIDE EFFECTS:
: 528 1516 1 file header updated, FCB updated
: 529 1517 1
: 530 1518 1 --
: 531 1519 1
: 532 1520 2 BEGIN
: 533 1521 2
: 534 1522 2 MAP
: 535 1523 2 HEADER : REF BBLOCK; ! file header arg
: 536 1524 2
: 537 1525 2 BIND_COMMON;
: 538 1526 2
: 539 1527 2 EXTERNAL ROUTINE
: 540 1528 2 INIT_FCB2 : L_NORM, ! initialize FCB
: 541 1529 2 ZERO_IDX : L_NORM, ! initialize directory index
: 542 1530 2 CHECKSUM : L_NORM, ! compute file header checksum
: 543 1531 2 WRITE_HEADER : L_NORM; ! write file header
: 544 1532 2
: 545 1533 2
: 546 1534 2 ! Update the end of file mark in the header and the FCB. Then adjust the
: 547 1535 2 ! directory index in the FCB. Finally checksum and write the file header.
: 548 1536 2 ! If this file header supports it, stuff the high water field to
: 549 1537 2 ! be the allocated size.
: 550 1538 2
: 551 1539 2
: 552 1540 2 BBLOCK [HEADER[FH2$W_RECATTR], FAT$W_EFBLKL] =
```



```

553      1541      2      .BBLOCK [HEADER[FH2$W_RECATTR], FAT$W_EFBLKL] + .DIRECTION;
554      1542
555      1543      2      IF .HEADER [FH2$B_IDOFFSET] GEQU ($BYTEOFFSET (FH2$L_HIGHWATER)+4)/2
556      1544      2      AND .HEADER [FH2$C_HIGHWATER] LSSU .BBLOCK [HEADER[FH2$W_RECATTR], FAT$W_EFBLKL]
557      1545      2      THEN
558      1546      2      HEADER [FH2$L_HIGHWATER] = .BBLOCK [HEADER[FH2$W_RECATTR], FAT$W_EFBLKL];
559      1547
560      1548      2      CLEANUP_FLAGS[CLF_NOBUILD] = 1;
561      1549      2      KERNEL_CALL (INIT_FCB2, .DIR_FCB, .HEADER);
562      1550      2      BBLOCK [HEADER[FH2$W_RECATTR], FAT$W_HIBLKL] = .DIR_FCB[FCB$L_FILESIZE];
563      1551      2      KERNEL_CALL (ZERO_IDX);
564      1552
565      1553      2      CHECKSUM (.HEADER);
566      1554      2      WRITE_HEADER ();
567      1555
568      1556      1      END;

```

```
.EXTRN  INIT_FCB2, ZERO_IDX
.EXTRN  CHECKSUM, WRITE_HEADER
```

```
0000 00000 FIX_HEADER:
```

				0000	00000	FIX-HEADER:				
			50	04	AC	DO	00002	.WORD	Save nothing	: 1484
		1E	A0	08	AC	A0	00006	MOVL	HEADER, R0	: 1540
			50	04	AC	DO	0000B	ADDW2	DIRECTION, 30(R0)	: 1541
			28		60	91	0000F	MOVL	HEADER, R0	: 1543
					0E	1F	00012	CMPB	(R0), #40	: :
4C	A0	1E	A0		00	ED	00014	BLSSU	1\$	: :
			10		05	1B	0001B	CMPZV	#0, #16, 30(R0), 76(R0)	: 1544
		4C	A0	1E	A0	3C	0001D	BLEQU	1\$	: :
		01	AA		08	88	00022	MOVZWL	30(R0), 76(R0)	: 1546
					04	AC	DD	BISB2	#8, 1(BASE)	: 1548
				00D0	CA	DD	00026	PUSHL	HEADER	: 1549
		0000G	CF		02	FB	0002D	PUSHL	208(BASE)	: :
			51	04	AC	DO	00032	CALLS	#2, INIT_FCB2	: :
			50	00D0	CA	DO	00036	MOVL	HEADER, R1	: 1550
		1A	A1	38	A0	B0	0003B	MOVL	208(BASE), R0	: :
		0000G	CF		00	FB	00040	MOVW	56(R0), 26(R1)	: :
				04	AC	DD	00045	CALLS	#0, ZERO_IDX	: 1551
		0000G	CF		01	FB	00048	PUSHL	HEADER	: 1553
		0000G	CF		00	FB	0004D	CALLS	#1, CHECKSUM	: :
					04	00052	CALLS	#0, WRITE_HEADER	: 1554	
							RET		: 1556	

; Routine Size: 83 bytes, Routine Base: \$CODE\$ + 03FB


```
570 1557 1 ROUTINE HANDLER (SIGNAL, MECHANISM) =
571 1558 1
572 1559 1 ++
573 1560 1
574 1561 1 FUNCTIONAL DESCRIPTION:
575 1562 1
576 1563 1 This routine is the condition handler for directory extension. It is
577 1564 1 enabled only during the truncate call (deallocating the old directory
578 1565 1 blocks). Normal error handling would cause the entire directory to
579 1566 1 be dropped on the floor. Since we already have a new good copy, we
580 1567 1 should forge ahead. Note that no error status is returned to the user,
581 1568 1 although we will log a system error.
582 1569 1
583 1570 1
584 1571 1 CALLING SEQUENCE:
585 1572 1 HANDLER (ARG1, ARG2)
586 1573 1
587 1574 1 INPUT PARAMETERS:
588 1575 1 ARG1: address of signal array
589 1576 1 ARG2: address of mechanism array
590 1577 1
591 1578 1 IMPLICIT INPUTS:
592 1579 1 FILE_HEADER: address of directory file header
593 1580 1
594 1581 1 OUTPUT PARAMETERS:
595 1582 1 NONE
596 1583 1
597 1584 1 IMPLICIT OUTPUTS:
598 1585 1 NONE
599 1586 1
600 1587 1 ROUTINE VALUE:
601 1588 1 $$$_RESIGNAL or none if unwind
602 1589 1
603 1590 1 SIDE EFFECTS:
604 1591 1 file header map area cleaned out
605 1592 1
606 1593 1 --
607 1594 1
608 1595 2 BEGIN
609 1596 2
610 1597 2 MAP
611 1598 2 SIGNAL : REF BBLOCK, ! signal array arg
612 1599 2 MECHANISM : REF BBLOCK; ! mechanism array arg
613 1600 2
614 1601 2
615 1602 2 ! Check the condition code for FCP error exit. Then initialize the header's
616 1603 2 ! map area and unwind. On other signals we simply resignal.
617 1604 2 !
618 1605 2
619 1606 2 IF .SIGNAL[CHFSL SIG_NAME] EQL $$$ CMODUSER
620 1607 2 THEN $UNWIND (DEPADR = MECHANISM[CHFSL_MCH_DEPTH]);
621 1608 2
622 1609 2 RETURN $$$_RESIGNAL; ! status is irrelevant if unwind
623 1610 2
624 1611 1 END; ! end of routine handler
```



```
                                .EXTRN  SYSSUNWIND
                                0000 00000 HANDLER: .WORD  Save nothing
                                AC D0 00002          MOVL   SIGNAL, R0
00000424 50 04 04 A0 D1 00006          CMPL   4(R0), #1060
                                0E 12 0000E          BNEQ   1$
                                7E D4 00010          CLRL   -(SP)
7E 08 AC 08 C1 00012          ADDL3  #8, MECHANISM, -(SP)
00000000G 00 02 FB 00017          CALLS  #2, SYSSUNWIND
50 0918 8F 3C 0001E 1$:      MOVZWL #2328, R0
                                04 00023          RET
                                : 1557
                                : 1606
                                : 1607
                                : 1609
                                : 1611
```

; Routine Size: 36 bytes, Routine Base: \$CODE\$ + 044E

```
: 625      1612 1
: 626      1613 1 END
: 627      1614 0 ELUDOM
```

PSECT SUMMARY

Name	Bytes	Attributes
\$CODE\$	1138	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	46	0	1000	00:01.9

COMMAND QUALIFIERS

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

```
: Size:      1134 code + 4 data bytes
: Run Time:   00:39.4
: Elapsed Time: 01:27.7
: Lines/CPU Min: 2459
: Lexemes/CPU-Min: 41809
: Memory Used: 394 pages
```


SHFDIR
V04-000

K 3
16-Sep-1984 01:10:06

VAX-11 Bliss-32 V4.0-742

Page 20

; Compilation Complete

0173 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

SCHFCB
LIS

SND5MB
LIS

SHFDLR
LIS

SNDERL
LIS

TRUNC
LIS

FAL

FAL
MAP

SELVOL
LIS

DAPDEF
MOL

SMALOC
LIS

SNOBAD
LIS

SWITUL
LIS

WITURN
LIS