

FFFFFFFFFFF	111	111	XXX	XXX
FFFFFFFFFFF	111	111	XXX	XXX
FFFFFFFFFFF	111	111	XXX	XXX
FFF	111111	111111	XXX	XXX
FFF	111111	111111	XXX	XXX
FFF	111111	111111	XXX	XXX
FFF	111	111		
FFF	111	111	XXX	XXX
FFF	111	111	XXX	XXX
FFFFFFFFF,FFF	111	111		
FFFFFFFFFFFFF	111	111	XXX	
FFFFFFFFFFFFF	111	111	XXX	
FFF	111	111		
FFF	111	111	XXX	XXX
FFF	111	111	XXX	XXX
FFF	111	111	XXX	XXX
FFF	111	111		
FFF	111	111		
FFF	111	111	XXX	XXX
FFF	111	111	XXX	XXX
FFF	111	111	XXX	XXX
FFF	111111111	111111111	XXX	XXX
FFF	111111111	111111111	XXX	XXX
FFF	111111111	111111111	XXX	XXX

Synt

 10C9
 10-C
 10-C
 10-C
 10-F
 10-S
 KICL

 KILL
 KILL
 LB_E
 LB_C
 LB_F
 LB_H
 LB_L
 LOCA
 LOCA
 LOCK

 LOCK
 LOCK
 LOCK
 LOC-
 LOC-
 L-CC
 L-CC
 L-DA
 L-DA
 MAIN
 MAKE
 MAKE
 MAKE
 MAKE
 MAKE
 MAKE

 MAKE
 MAKE
 MAP-
 MAP-

 MAP
 MARI
 MARI
 MARI
 MARI
 MARI

```

AAAAAA  CCCCCC  PPPPPPP  CCCCCC  NN  NN  TTTTTTTTTT  RRRRRRRR  LL
AAAAAA  CCCCCC  PPPPPPP  CCCCCC  NN  NN  TTTTTTTTTT  RRRRRRRR  LL
AA      AA  CC      PP      PP  CC      NN  NN  TT      RR      LL
AA      AA  CC      PP      PP  CC      NN  NN  TT      RR      LL
AA      AA  CC      PP      PP  CC      NN  NN  TT      RR      LL
AA      AA  CC      PP      PP  CC      NN  NN  TT      RR      LL
AA      AA  CC      PP      PP  CC      NN  NN  TT      RR      LL
AAAAAAAA  CC      PPPPPPP  CC      NN  NN  TT      RRRRRRRR  LL
AAAAAAAA  CC      PPPPPPP  CC      NN  NN  TT      RRRRRRRR  LL
AA      AA  CC      PP      PP  CC      NN  NN  TT      RR  RR  LL
AA      AA  CC      PP      PP  CC      NN  NN  TT      RR  RR  LL
AA      AA  CC      PP      PP  CC      NN  NN  TT      RR  RR  LL
AA      AA  CCCCCC  PP      CCCCCC  NN  NN  TT      RR  RR  LLLLLLLLLL
AA      AA  CCCCCC  PP      CCCCCC  NN  NN  TT      RR  RR  LLLLLLLLLL

```

```

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
LLLLLLLL  IIIIII  SSSSSSSS
LLLLLLLL  IIIIII  SSSSSSSS

```

```
1 0001 0 MODULE ACPCNTRL (
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 module implements the ACP control I/O function.
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: 23-May-1979 17:07
48 0048 1
49 0049 1 MODIFIED BY:
50 0050 1
51 0051 1 V03-020 CDS0010 Christian D. Saether 4-Aug-1984
52 0052 1 Fix a bug in lock_volume which can leave current_ucb
53 0053 1 zero on incomplete volume sets.
54 0054 1 Modify l_map_pointer linkage to declare notused registers.
55 0055 1
56 0056 1 V03-019 ACG0438 Andrew C. Goldstein, 18-Jul-1984 19:21
57 0057 1 Add cache flush control function; use central routine
```

58	0058	1	for all simple \$DEQ's.
59	0059	1	
60	0060	1	V03-018 CWH3018 CW Hobbs 27-Jul-1984
61	0061	1	Fix broken branch.
62	0062	1	
63	0063	1	V03-017 CDS0009 Christian D. Saether 8-May-1984
64	0064	1	Set NOALLOC on lock_vol function to prevent the
65	0065	1	process that holds the lock from modifying the
66	0066	1	volume structure itself.
67	0067	1	
68	0068	1	V03-016 CDS0008 Christian D. Saether 27-Dec-1983
69	0069	1	Use BIND_COMMON macro. Adjust routine and external
70	0070	1	declarations.
71	0071	1	
72	0072	1	V03-015 CDS0007 Christian D. Saether 18-Oct-1983
73	0073	1	Clear NOALLOC on unlock to be more compatible with
74	0074	1	previous behavior.
75	0075	1	
76	0076	1	V03-014 CDS0006 Christian D. Saether 17-Oct-1983
77	0077	1	Do request block checking here except for null
78	0078	1	control function.
79	0079	1	Also update volume free space in value block on
80	0080	1	unlock function to correctly propagate through cluster.
81	0081	1	
82	0082	1	V03-013 CDS0005 Christian D. Saether 14-Oct-1983
83	0083	1	LOCK_VOL now blocks all volume modification activity
84	0084	1	instead of setting NOALLOC flag.
85	0085	1	
86	0086	1	V03-012 CDS0004 Christian D. Saether 3-Oct-1983
87	0087	1	Take out allocation lock prior to reading SCB.
88	0088	1	
89	0089	1	V03-011 CDS0003 Christian D. Saether 14-Sep-1983
90	0090	1	Modify interface to SERIAL_FILE.
91	0091	1	
92	0092	1	V03-010 CDS0002 Christian D. Saether 5-May-1983
93	0093	1	Interlock processing on REMAP_FILE call with
94	0094	1	SERIAL_FILE routine.
95	0095	1	
96	0096	1	V03-009 CDS0001 Christian D. Saether 6-Jan-1983
97	0097	1	Changes to accomodate cluster dismount. Decrement
98	0098	1	write counter in the SCB.
99	0099	1	
100	0100	1	V03-008 LMP0027 L. Mark Pilant, 18-May-1982 11:20
101	0101	1	Rearrange some code sequences to avoid the possibility of
102	0102	1	taking a page fault at an elevated IPL.
103	0103	1	
104	0104	1	V03-007 ACG0285 Andrew C. Goldstein, 12-Apr-1982 17:24
105	0105	1	Fix cathedral window logic for empty headers,
106	0106	1	do not unlock volume if there is no storage map
107	0107	1	
108	0108	1	V03-006 LMP0019 L. Mark Pilant, 1-Apr-1982 14:20
109	0109	1	Correct a boundary condition that would cause windows to
110	0110	1	be corrupted.
111	0111	1	
112	0112	1	V03-001 LMP0016 L. Mark Pilant, 25-Mar-1982 13:35
113	0113	1	Remove diddling of the COMPLETE bit in the window segments.
114	0114	1	Also, fix a bug that caused the system to crash if a remap

```

115      0115 1 | was forced on a file whose corresponding window did not start
116      0116 1 | with VBN 1.
117      0117 1 |
118      0118 1 | V02-004 LMP0012 L. Mark Pilant, 15-Mar-1982 16:00
119      0119 1 | Add a routine to set WCB$M_COMPLETE in all the window
120      0120 1 | segments if the remap succeeds.
121      0121 1 |
122      0122 1 | V02-003 LMP0003 L. Mark Pilant, 25-Nov-1981 11:40
123      0123 1 | Add routine to remap a file into multiple windows.
124      0124 1 |
125      0125 1 | V02-002 ACG0167 Andrew C. Goldstein, 16-Apr-1980 19:24
126      0126 1 | Previous revision history moved to F11B.REV
127      0127 1 | **
128      0128 1 |
129      0129 1 |
130      0130 1 | LIBRARY 'SYSS$LIBRARY:LIB.L32';
131      0131 1 | REQUIRE 'SRC$:FCPDEF.B32';
132      1122 1 |
133      1123 1 | | Range of control function codes recognized by this module.
134      1124 1 | |
135      1125 1 | |
136      1126 1 | |
137      1127 1 | LITERAL
138      1128 1 | MIN_CNTRLFUNC = MINU (FIB$C_LOCK_VOL,
139      1129 1 | FIB$C_UNLK_VOL,
140      1130 1 | FIB$C_ENA_QUOTA,
141      1131 1 | FIB$C_ADD_QUOTA,
142      1132 1 | FIB$C_EXA_QUOTA,
143      1133 1 | FIB$C_MOD_QUOTA,
144      1134 1 | FIB$C_REM_QUOTA,
145      1135 1 | FIB$C_DSA_QUOTA,
146      1136 1 | FIB$C_REMAP,
147      1137 1 | FIB$C_FLUSH_CACHE
148      1138 1 | ),
149      1139 1 |
150      1140 1 | MAX_CNTRLFUNC = MAXU (FIB$C_LOCK_VOL,
151      1141 1 | FIB$C_UNLK_VOL,
152      1142 1 | FIB$C_ENA_QUOTA,
153      1143 1 | FIB$C_ADD_QUOTA,
154      1144 1 | FIB$C_EXA_QUOTA,
155      1145 1 | FIB$C_MOD_QUOTA,
156      1146 1 | FIB$C_REM_QUOTA,
157      1147 1 | FIB$C_DSA_QUOTA,
158      1148 1 | FIB$C_REMAP,
159      1149 1 | FIB$C_FLUSH_CACHE
160      1150 1 | ),
161      1151 1 |
162      1152 1 | MIN_CACHE_CODE = MINU (FIB$C_FID_CACHE,
163      1153 1 | FIB$C_EXTENT_CACHE,
164      1154 1 | FIB$C_QUOTA_CACHE
165      1155 1 | ),
166      1156 1 |
167      1157 1 | MAX_CACHE_CODE = MAXU (FIB$C_FID_CACHE,
168      1158 1 | FIB$C_EXTENT_CACHE,
169      1159 1 | FIB$C_QUOTA_CACHE
170      1160 1 | );
171      1161 1 |
```

ACPCNTRL
V04-000

K 16
15-Sep-1984 23:52:53 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:30:08 [F11X.SRC]ACPCNTRL.B32;1

Page 4
(1)

:	172	1162	1	FORWARD ROUTINE	:		:	
:	173	1163	1	ACPCNTRL	:	L_NORM,	:	ACPCNTRL function routine
:	174	1164	1	DISMOUNT	:	L_NORM,	:	do volume dismount processing
:	175	1165	1	LOCK_VOLUME	:	L_NORM,	:	lock or unlock volume
:	176	1166	1	MARK_CATHEDRAL	:	NOVALUE,	:	flag window as being cathedral
:	177	1167	1	ADD_WINDOW	:	NOVALUE,	:	add a window to the queue
:	178	1168	1	REMOVE_WINDOW	:	NOVALUE,	:	remove and deallocate a window segment
:	179	1169	1	LAST_SEGMENT	:	NOVALUE;	:	set the window as the last segment

```
181 1170 1 GLOBAL ROUTINE ACPCONTROL : L_NORM =
182 1171 1
183 1172 1 ++
184 1173 1
185 1174 1 FUNCTIONAL DESCRIPTION:
186 1175 1
187 1176 1 This routine implements the ACP control I/O function. It sets up
188 1177 1 context and dispatches on the control function.
189 1178 1
190 1179 1 CALLING SEQUENCE:
191 1180 1 ACPCONTROL ()
192 1181 1
193 1182 1 INPUT PARAMETERS:
194 1183 1 NONE
195 1184 1
196 1185 1 IMPLICIT INPUTS:
197 1186 1 CLEANUP_FLAGS: cleanup action and status flags
198 1187 1 IO_PACKET: address of I/O request packet
199 1188 1
200 1189 1 OUTPUT PARAMETERS:
201 1190 1 NONE
202 1191 1
203 1192 1 IMPLICIT OUTPUTS:
204 1193 1 NONE
205 1194 1
206 1195 1 ROUTINE VALUE:
207 1196 1 assorted status values
208 1197 1
209 1198 1 SIDE EFFECTS:
210 1199 1 control function executed
211 1200 1
212 1201 1 --
213 1202 1
214 1203 2 BEGIN
215 1204 2
216 1205 2 LOCAL
217 1206 2 FIB : REF BBLOCK, ! address of user FIB
218 1207 2 ABD : REF BBLOCK, ! address of buffer descriptor
219 1208 2 STATUS; ! return status from called routine
220 1209 2
221 1210 2 BIND_COMMON;
222 1211 2
223 1212 2 EXTERNAL ROUTINE
224 1213 2 START_REQUEST : L_JSB ADDRESSING MODE (GENERAL), ! request blocking check
225 1214 2 SERIAL_FILE : L_NORM, ! serialize file processing
226 1215 2 GET_FIB : L_NORM, ! get user FIB
227 1216 2 REMAP_FILE : L_NORM NOVALUE, ! remap the file into segmented windows
228 1217 2 QUOTA_FILE_OP : L_NORM, ! quota file maint operation
229 1218 2 CONN_QFILE : L_NORM, ! connect quota file
230 1219 2 ALLOCATION_LOCK : L_NORM, ! acquire volume allocation lock
231 1220 2 DELETE_FID : L_NORM, ! free file ID or flush file ID cache
232 1221 2 PURGE_EXTENT : L_NORM, ! purge out extent cache
233 1222 2 FLUSH_QUO_CACHE : L_NORM; ! flush out quota cache
234 1223 2
235 1224 2
236 1225 2 ! Set up control block pointers. If there is no complex buffer packet, then
237 1226 2 ! this is an I/O kill call, which is a NOP.
```

```
238 1227 2 !
239 1228 2
240 1229 2 IF NOT .IO_PACKET[IRPSV_COMPLX] THEN RETURN 1;
241 1230 2
242 1231 2 ABD = .BBLOCK [.IO_PACKET[IRPSL_SVAPTE], AIBSL_DESCRIPTOR];
243 1232 2 FIB = GET_FIB (.ABD);
244 1233 2
245 1234 2 IF .FIB[FIBSW_CNTRLFUNC] EQL 0
246 1235 2     AND NOT .BBLOCK [IO_PACKET[IRPSW_FUNC], IOSV_DMOUNT]
247 1236 2 THEN RETURN 1;
248 1237 2 ! 0 is a NOP
249 1238 2 ! Case on the control function and stall for the volume blocking lock
250 1239 2 as necessary.
251 1240 2
252 1241 2
253 1242 2 CASE .FIB[FIBSW_CNTRLFUNC] FROM MIN_CNTRLFUNC TO MAX_CNTRLFUNC OF
254 1243 2     SET
255 1244 2
256 1245 2     [FIBSC_ADD_QUOTA]: BEGIN
257 1246 2         IF .BLOCK_LOCKID EQL 0
258 1247 2         THEN
259 1248 2             BEGIN
260 1249 2                 START_REQUEST ();
261 1250 2                 BLOCK_CHECK = 1;
262 1251 2             END;
263 1252 2         END;
264 1253 2
265 1254 2     [INRANGE, OUTFRANGE]: 0;
266 1255 2
267 1256 2     TES;
268 1257 2
269 1258 2 IF .BBLOCK [IO_PACKET[IRPSW_FUNC], IOSV_DMOUNT]
270 1259 2 THEN RETURN DISMOUNT ();
271 1260 2
272 1261 2 ! Dispatch on the control function.
273 1262 2 !
274 1263 2
275 1264 2 CASE .FIB[FIBSW_CNTRLFUNC] FROM MIN_CNTRLFUNC TO MAX_CNTRLFUNC OF
276 1265 2     SET
277 1266 2
278 1267 2     [FIBSC_REMAP]: BEGIN
279 1268 2         LOCAL
280 1269 2             FCB : REF BBLOCK;
281 1270 2
282 1271 2             FCB = .CURRENT_WINDOW [WCB$FID];
283 1272 2
284 1273 2             PRIM_LCKINDX = SERIAL_FILE (FCB [FCB$FID]);
285 1274 2
286 1275 2             REMAP_FILE ();
287 1276 2         END;
288 1277 2
289 1278 2     [FIBSC_LOCK_VOL,
290 1279 2     FIBSC_UNLK_VOL]: BEGIN
291 1280 2         IF NOT .CLEANUP_FLAGS[CLF_SYSPRV]
292 1281 2         THEN ERR_EXIT (SS$NOPRIV);
293 1282 2         KERNEL_CALL (LOCK_VOLUME, .FIB[FIBSW_CNTRLFUNC]);
294 1283 2     END;
```

```

295 1284 2
296 1285 2
297 1286 2
298 1287 2
299 1288 2
300 1289 2
301 1290 2
302 1291 2
303 1292 2
304 1293 2
305 1294 2
306 1295 2
307 1296 2
308 1297 2
309 1298 2
310 1299 2
311 1300 2
312 1301 2
313 1302 2
314 1303 2
315 1304 2
316 1305 2
317 1306 2
318 1307 2
319 1308 2
320 1309 2
321 1310 2
322 1311 2
323 1312 2
324 1313 2
325 1314 2
326 1315 2
327 1316 2
328 1317 2
329 1318 2
330 1319 1

[FIBSC_ENA_QUOTA]: CONN_QFILE (.ABD, .FIB);

[FIBSC_ADD_QUOTA,
 FIBSC_EXA_QUOTA,
 FIBSC_MOD_QUOTA,
 FIBSC_REM_QUOTA,
 FIBSC_DSA_QUOTA]: QUOTA_FILE_OP (.ABD, .FIB);

[FIBSC_FLUSH_CACHE]:
    BEGIN
    ALLOCATION_LOCK ();
    CASE .FIB[FIBSL_CNTRLVAL]
    FROM MIN_CACHE_CODE TO MAX_CACHE_CODE OF
        SET
            [FIBSC_FID_CACHE]: DELETE_FID (0);
            [FIBSC_EXTENT_CACHE]: PURGE_EXTENT (0,0);
            [FIBSC_QUOTA_CACHE]: FLUSH_QUO_CACHE ();
            [INRANGE,
             OUTRANGE]:      ERR_EXIT (SS$_ILLIOFUNC);
        TES;
    END;

[INRANGE,
 OUTRANGE]:      ERR_EXIT (SS$_ILLIOFUNC);
TES;
RETURN 1;
END;
```

! end of routine ACPCNTRL

```

.TITLE ACPCNTRL
.IDENT \V04-000\

.EXTRN START_REQUEST, SERIAL_FILE
.EXTRN GET_FIB, REMAP_FILE
.EXTRN QUOTA_FILE_OP, CONN_QFILE
.EXTRN ALLOCATION_LOCK
.EXTRN DELETE_FID, PURGE_EXTENT
.EXTRN FLUSH_QUO_CACHE
```

.PSECT \$CODE\$,NOWRT,2

```

.ENTRY ACPCNTRL, Save R2,R3
MOVL -112(BASE), R0
BBS #3, 42(R0), 2$
BRW 20$
MOVL @44(R0), ABD
PUSHL ABD
CALLS #1, GET_FIB
```

```

03      2A      50      90      AA      DO      00002
          A0      03      EO      00006
          53      2C      00E8    31      0000B 1$:
          0000G CF      80      DO      0000E 2$:
          53      DD      00012
          01      FB      00014
```

```

: 1170
: 1229
:
: 1231
: 1232
:
```

[illegible]

			37	11	000BD		BRB	20\$		
			52	DD	000BF	13\$:	PUSHL	FIB		1291
			53	DD	000C1		PUSHL	ABD		
	0000G	CF	02	FB	000C3		CALLS	#2, QUOTA_FILE_OP		
			2C	11	000C8		BRB	20\$		
	0000G	CF	00	FB	000CA	14\$:	CALLS	#0, ALLOCATION_LOCK		1295
02		01	18	A2	CF	000CF	CASEL	24(FIB), #1, #2		1296
001D	0014			000B		000D4	15\$:	.WORD	17\$-15\$,- 18\$-15\$,- 19\$-15\$	
			00F4	8F	BF	000DA	16\$:	CHMU	#244	1307
				04	000DE		RET			
	0000G	CF	7E	D4	000DF	17\$:	CLRL	-(SP)		1300
			01	FB	000E1		CALLS	#1, DELETE_FID		
			0E	11	000E6		BRB	20\$		
	0000G	CF	7E	7C	000E8	18\$:	CLRQ	-(SP)		1302
			02	FB	000EA		CALLS	#2, PURGE_EXTENT		
	0000G	CF	05	11	000EF		BRB	20\$		
			00	FB	000F1	19\$:	CALLS	#0, FLUSH_QUO_CACHE		1304
		50	01	D0	000F6	20\$:	MOVL	#1, R0		1317
				04	000F9		RET			1319

; Routine Size: 250 bytes, Routine Base: \$CODE\$ + 0000

```

332 1320 1 ROUTINE DISMOUNT : L_NORM =
333 1321 1
334 1322 1 ++
335 1323 1
336 1324 1 FUNCTIONAL DESCRIPTION:
337 1325 1
338 1326 1 This routine handles the DISMOUNT subfunction of the ACPCNTRL
339 1327 1 function. It reads the volume's storage control block, clears the
340 1328 1 status bits, and rewrites the block to indicate that the volume
341 1329 1 has been properly dismounted. (Once a volume is marked for dismount,
342 1330 1 all deferred write cacheing, etc., is turned off.)
343 1331 1 The file ID, extent, and quota caches are also flushed.
344 1332 1
345 1333 1
346 1334 1 CALLING SEQUENCE:
347 1335 1 DISMOUNT ()
348 1336 1
349 1337 1 INPUT PARAMETERS:
350 1338 1 NONE
351 1339 1
352 1340 1 IMPLICIT INPUTS:
353 1341 1 CURRENT_VCB: VCB of volume
354 1342 1 CURRENT_UCB: UCB of volume
355 1343 1
356 1344 1 OUTPUT PARAMETERS:
357 1345 1 NONE
358 1346 1
359 1347 1 IMPLICIT OUTPUTS:
360 1348 1 NONE
361 1349 1
362 1350 1 ROUTINE VALUE:
363 1351 1 1 if success
364 1352 1
365 1353 1 SIDE EFFECTS:
366 1354 1 storage control block rewritten
367 1355 1
368 1356 1 --
369 1357 1
370 1358 2 BEGIN
371 1359 2
372 1360 2 LOCAL
373 1361 2 SCB : REF BBLOCK; ! buffer containing storage control block
374 1362 2
375 1363 2 BIND_COMMON;
376 1364 2
377 1365 2 EXTERNAL ROUTINE
378 1366 2 ALLOCATION_LOCK : L_NORM, ! synchronize allocation/deallocation
379 1367 2 DELETE_FID : L_NORM, ! free file ID or flush file ID cache
380 1368 2 PURGE_EXTENT : L_NORM, ! purge out extent cache
381 1369 2 FLUSH_QUO_CACHE : L_NORM, ! flush out quota cache
382 1370 2 READ_BLOCK : L_NORM, ! read a disk block
383 1371 2 WRITE_BLOCK : L_NORM, ! write a disk block
384 1372 2 CHECKSUM : L_NORM, ! compute block checksum
385 1373 2
386 1374 2
387 1375 2 ! Flush the caches and clear the SCB flags if the volume is not write
388 1376 2 ! locked or allocation locked.
```

```

389      1377 2 !
390      1378
391      1379 IF NOT (.CURRENT_VCB[VCB$V_NOALLOC]
392      1380         OR .BBLOCK [CURRENT_UCB[UCB$L_DEVCHAR], DEV$V_SWL])
393      1381 THEN
394      1382     BEGIN
395      1383     ALLOCATION_LOCK ();
396      1384     DELETE_FID (0);
397      1385     PURGE_EXTENT (0, 0);
398      1386     FLUSH_QUO_CACHE ();
399      1387
400      1388     SCB = READ_BLOCK (.CURRENT_VCB[VCB$L_SBMAPLBN] - 1, 1, BITMAP_TYPE);
401      1389     SCB [SCB$W_WRITECNT] = .SCB [SCB$W_WRITECNT] - 1;
402      1390     IF .SCB [SCB$W_WRITECNT] EQL 0
403      1391     THEN
404      1392         SCB[SCB$L_STATUS] = 0;
405      1393     CHECKSUM (.SCB);
406      1394     WRITE_BLOCK (.SCB);
407      1395     END;
408      1396
409      1397 2 1
410      1398 1 END;

```

! end of routine DISMOUNT

```

.EXTRN READ_BLOCK, WRITE_BLOCK
.EXTRN CHECKSUM

```

0004 0000 DISMOUNT:

		50	98	AA	D0	00002	.WORD	Save R2	1320
		A0		04	E0	00006	MOVL	-104(BASE), R0	1379
4C	0B	50	94	AA	D0	0000B	BBS	#4, 11(R0), 2\$	
		A0		01	E0	0000F	MOVL	-108(BASE), R0	1380
43	3B	CF		00	FB	00014	BBS	#1, 59(R0), 2\$	
	0000G			7E	D4	00019	CALLS	#0, ALLOCATION_LOCK	1383
		CF		01	FB	0001B	CLRL	-(SP)	1384
	0000G			7E	7C	00020	CALLS	#1, DELETE_FID	
		CF		02	FB	00022	CLRL	-(SP)	1385
	0000G			00	FB	00027	CALLS	#2, PURGE_EXTENT	
		CF		01	DD	0002C	CALLS	#0, FLUSH_QUO_CACHE	1386
				01	DD	0002E	PUSHL	#1	1388
		50	98	AA	D0	00030	PUSHL	#1	
7E	34	A0		01	C3	00034	MOVL	-104(BASE), R0	
	0000G	CF		03	FB	00039	SUBL3	#1, 52(R0), -(SP)	
		52		50	D0	0003E	CALLS	#3, READ_BLOCK	
			20	A2	B7	00041	MOVL	R0, SCB	
				03	12	00044	DECW	32(SCB)	1389
			18	A2	D4	00046	BNEQ	1\$	1390
				52	DD	00049	CLRL	24(SCB)	1392
	0000G	CF		01	FB	0004B	PUSHL	SCB	1393
				52	DD	00050	CALLS	#1, CHECKSUM	
	0000G	CF		01	FB	00052	PUSHL	SCB	1394
		50		01	D0	00057	CALLS	#1, WRITE_BLOCK	
				04	0005A		MOVL	#1, R0	1398
							RET		

; Routine Size: 91 bytes, Routine Base: \$CODE\$ + 00FA

ACPCNTRL
V04-000

6 1
15-Sep-1984 23:52:53
14-Sep-1984 12:30:08

VAX-11 Bliss-32 V4.0-742
[F11X.SRC]ACPCNTRL.B32;1

Page 12
(3)

AC
VO

```
412 1399 1 GLOBAL ROUTINE REMAP_FILE : NOVALUE L_NORM =
413 1400 1
414 1401 1 ++
415 1402 1
416 1403 1 FUNCTIONAL DESCRIPTION:
417 1404 1
418 1405 1 This routine is called when it becomes necessary to guarantee that
419 1406 1 the entire file is mapped. This is done by creating, if necessary,
420 1407 1 multiple WCB's and linking them together.
421 1408 1
422 1409 1 CALLING SEQUENCE:
423 1410 1 REMAP_FILE ()
424 1411 1
425 1412 1 INPUT PARAMETERS:
426 1413 1 none
427 1414 1
428 1415 1 IMPLICIT INPUTS:
429 1416 1 PRIMARY_FCB: address of the current primary FCB
430 1417 1 CURRENT_WINDOW: address of the current primary window segment
431 1418 1
432 1419 1 OUTPUT PARAMETERS:
433 1420 1 none
434 1421 1
435 1422 1 IMPLICIT OUTPUTS:
436 1423 1 none
437 1424 1
438 1425 1 ROUTINE VALUE:
439 1426 1 none
440 1427 1
441 1428 1 SIDE EFFECTS:
442 1429 1 As many WCB's as are needed are allocated and linked to provide
443 1430 1 mapping for the entire file. Any errors are noted for the user.
444 1431 1
445 1432 1 --
446 1433 1
447 1434 2 BEGIN
448 1435 2
449 1436 2 LINKAGE
450 1437 2 L_MAP_POINTER A = JSB:
451 1438 2 GLOBAL (HEADER_COUNT = 6, HDR_LBN = 7, HEADER_POINTER = 8)
452 1439 2 NOTUSED (2,3,4,5,9,10,11);
453 1440 2
454 1441 2 GLOBAL REGISTER
455 1442 2 HEADER_COUNT = 6, ! retrieval pointer count
456 1443 2 HDR_LBN = 7, ! retrieval pointer start LBN
457 1444 2 HEADER_POINTER = 8 : REF BBLOCK; ! pointer into map area
458 1445 2
459 1446 2 LABEL
460 1447 2 HEADER_CHECK, ! loop to check window/header correspondence
461 1448 2 WINDOW_TRUNCATE; ! loop to match up last FCB with a window
462 1449 2 LOCAL
463 1450 2 WINDOW_SEGMENT : REF BBLOCK, ! address of the next window segment
464 1451 2 OLD_WINDOW : REF BBLOCK, ! the original window
465 1452 2 NEW_WINDOW : REF BBLOCK, ! the new window list
466 1453 2 FCB : REF BBLOCK, ! address of the current FCB
467 1454 2 LAST_FCB : REF BBLOCK, ! address of the last FCB
468 1455 2 HEADER : REF BBLOCK, ! address of the header owned by an FCB
```

```

469      1456 2      HEADER VBN,      ! current VBN in the header
470      1457 2      NEXT_SEGMENT : REF BBLOCK, ! address of the segment after the next
471      1458 2      WINDOW_POINTER : REF BBLOCK, ! address of the window map area
472      1459 2      WINDOW_VBN,      ! current VBN in the window
473      1460 2      WINDOW_ENDVBN;    ! ending VBN of the window
474      1461 2
475      1462 2 BIND_COMMON;
476      1463 2
477      1464 2 EXTERNAL ROUTINE
478      1465 2      DEALLOCATE : L_NORM,      ! deallocate a block of memory
479      1466 2      READ_HEADER : L_NORM,      ! read a file header specified by FCB
480      1467 2      TURN_WINDOW : L_NORM,      ! map the file header specified
481      1468 2      MARK_COMPLETE : L_NORM NOVALUE, ! mark all windows as complete
482      1469 2      MARK_INCOMPLETE : L_NORM;    ! mark all window segments as incomplete
483      1470 2
484      1471 2 EXTERNAL ROUTINE
485      1472 2      GET_MAP_POINTER : L_MAP_POINTER_A; ! get next pointer from header
486      1473 2
487      1474 2 ! Make sure that a file is there.
488      1475 2 !
489      1476 2
490      1477 2 IF .CURRENT_WINDOW EQL 0 THEN ERR_EXIT (SS$_FILNOTACC);
491      1478 2
492      1479 2 ! Make sure it is actually necessary to do the remap operation.
493      1480 2 !
494      1481 2
495      1482 2 IF .CURRENT_WINDOW[WCB$V_COMPLETE]
496      1483 2 AND .CURRENT_WINDOW[WCB$V_CATHEDRAL]
497      1484 2 THEN RETURN;
498      1485 2
499      1486 2 ! If there is a file accessed, try to build any necessary window segments.
500      1487 2 ! There are three cases which can arise in trying to remap the entire file.
501      1488 2 ! 1) The window completely maps the file but it was not required to; in this
502      1489 2 ! case it is simply necessary to set WCB$V_CATHEDRAL. 2) The window was
503      1490 2 ! previously complete, but is no longer due to an extension of the file; in
504      1491 2 ! this case it is necessary to add the new window pointers to the last window
505      1492 2 ! segment (which may be the primary window). 3) The file was never completely
506      1493 2 ! mapped. In this case there are no special special conditions to consider.
507      1494 2 ! All that is necessary is to traverse the linked FCB's to build the window
508      1495 2 ! segments.
509      1496 2 !
510      1497 2
511      1498 2 ! First case; WCB$V_COMPLETE is set. Simply set WCB$V_CATHEDRAL and return.
512      1499 2 !
513      1500 2
514      1501 2 IF .CURRENT_WINDOW[WCB$V_COMPLETE] AND NOT .CURRENT_WINDOW[WCB$V_CATHEDRAL]
515      1502 2 THEN
516      1503 2     BEGIN
517      1504 2     KERNEL_CALL (MARK_CATHEDRAL, .CURRENT_WINDOW);
518      1505 2     RETURN;
519      1506 2     END;
520      1507 2
521      1508 2 ! Second case; the file was previously mapped complete. Locate the FCB which
522      1509 2 ! corresponds to the last window segment and start adding from there.
523      1510 2 !
524      1511 2
525      1512 2 IF .CURRENT_WINDOW[WCB$V_CATHEDRAL]
```

```
526 1513 2 THEN
527 1514 3 BEGIN
528 1515 3 WINDOW_SEGMENT = .CURRENT_WINDOW;
529 1516 3 FCB = .PRIMARY_FCB;
530 1517 3
531 1518 3 UNTIL .WINDOW_SEGMENT[WCB$LINK] EQL 0
532 1519 3 DO WINDOW_SEGMENT = .WINDOW_SEGMENT[WCB$LINK];
533 1520 3 NEW_WINDOW = .WINDOW_SEGMENT; ! remember current end point
534 1521 3
535 1522 3 WINDOW_ENDVBN = .WINDOW_SEGMENT[WCB$STVBN];
536 1523 3 WINDOW_POINTER = .WINDOW_SEGMENT + WCB$C_MAP;
537 1524 3 DECR J FROM .WINDOW_SEGMENT[WCB$NMAP] TO 1 DO
538 1525 4 BEGIN
539 1526 4 WINDOW_ENDVBN = .WINDOW_ENDVBN + .WINDOW_POINTER[WCB$W_COUNT];
540 1527 4 WINDOW_POINTER = .WINDOW_POINTER + 6;
541 1528 3 END;
542 1529 3
543 1530 4 HEADER_CHECK: BEGIN
544 1531 4 LAST_FCB = .FCB; ! in case only 1 FCB
545 1532 4 DO
546 1533 5 BEGIN
547 1534 5 IF .FCB[FCB$STVBN] GTR .WINDOW_ENDVBN THEN EXITLOOP 0;
548 1535 5 LAST_FCB = .FCB;
549 1536 5 FCB = .FCB[FCB$EXFCB];
550 1537 5 END
551 1538 4 UNTIL .FCB EQL 0;
552 1539 4 FCB = .LAST_FCB;
553 1540 4 HEADER = READ_HEADER (0, .FCB);
554 1541 4 HEADER_VBN = .FCB[FCB$STVBN];
555 1542 4 HEADER_POINTER = .HEADER + .HEADER[FH2$B_MPOFFSET]*2;
556 1543 4 IF .WINDOW_ENDVBN EQL .HEADER_VBN THEN LEAVE HEADER_CHECK;
557 1544 4 UNTIL .HEADER_POINTER GEQA .HEADER + (.HEADER[FH2$B_MPOFFSET] + .HEADER[FH2$B_MAP_INUSE]) * 2
558 1545 4 DO
559 1546 5 BEGIN
560 1547 5 GET_MAP_POINTER ();
561 1548 5 IF .WINDOW_ENDVBN GEQ .HEADER_VBN
562 1549 5 AND .WINDOW_ENDVBN LSS .HEADER_VBN + .HEADER_COUNT
563 1550 5 THEN LEAVE HEADER_CHECK;
564 1551 5 HEADER_VBN = .HEADER_VBN + .HEADER_COUNT;
565 1552 4 END;
566 1553 4 FCB = .FCB[FCB$EXFCB];
567 1554 4
568 1555 4 ! The last VBN mapped does not have a corresponding FCB. In this case it
569 1556 4 ! is necessary to locate the window segment that corresponds to the last
570 1557 4 ! FCB.
571 1558 4
572 1559 4
573 1560 4 WINDOW_SEGMENT = .CURRENT_WINDOW;
574 1561 5 WINDOW_TRUNCATE: BEGIN
575 1562 5 DO
576 1563 6 BEGIN
577 1564 6 WINDOW_VBN = .WINDOW_SEGMENT[WCB$STVBN];
578 1565 6 IF .WINDOW_VBN EQL .HEADER_VBN THEN LEAVE WINDOW_TRUNCATE;
579 1566 6 WINDOW_POINTER = .WINDOW_SEGMENT + WCB$C_MAP;
580 1567 6 DECR J FROM .WINDOW_SEGMENT[WCB$NMAP] TO 1 DO
581 1568 7 BEGIN
582 1569 7 WINDOW_VBN = .WINDOW_VBN + .WINDOW_POINTER[WCB$W_COUNT];
```

```
583      1570 7      WINDOW_POINTER = .WINDOW_POINTER + 6;
584      1571 7      IF .WINDOW_VBN EQL .HEADER_VBN THEN LEAVE WINDOW_TRUNCATE;
585      1572 6      END;
586      1573 6      WINDOW_SEGMENT = .WINDOW_SEGMENT[WCB$$_LINK];
587      1574 6      END
588      1575 5      UNTIL .WINDOW_SEGMENT EQL 0;
589      1576 5      BUG_CHECK (WCBFCBMNG, FATAL 'WCB/FCB correspondence broken');
590      1577 5
591      1578 5      END;
592      1579 4      ! end of block WINDOW_TRUNCATE
593      1580 4
594      1581 4      ! The window which corresponds to the last FCB has been found. Truncate the
595      1582 4      ! current window and remove any succeeding window segments.
596      1583 4
597      1584 4
598      1585 4      FCB = .LAST_FCB;
599      1586 4      NEXT_SEGMENT = .WINDOW_SEGMENT[WCB$$_LINK];
600      1587 4      KERNEL_CALL (LAST_SEGMENT, .WINDOW_SEGMENT);
601      1588 4      UNTIL .NEXT_SEGMENT EQL 0
602      1589 4      DO
603      1590 5      BEGIN
604      1591 5      LOCAL JUNK_SEGMENT : REF BBLOCK; ! address of block to deallocate
605      1592 5      JUNK_SEGMENT = .NEXT_SEGMENT;
606      1593 5      NEXT_SEGMENT = .NEXT_SEGMENT[WCB$$_LINK];
607      1594 5      KERNEL_CALL (REMOVE_WINDOW, .JUNK_SEGMENT);
608      1595 4      END;
609      1596 4
610      1597 3      END;
611      1598 3      ! end of block HEADER_CHECK
612      1599 3
613      1600 3      ! Map any additional file headers or rebuild the last window if cleaning up
614      1601 3      ! from an extend operation.
615      1602 3
616      1603 3      WHILE 1 DO
617      1604 4      BEGIN
618      1605 4      KERNEL_CALL (TURN_WINDOW, .WINDOW_SEGMENT, .HEADER, 1, .FCB[FCB$$_STVBN]);
619      1606 4      IF .CLEANUP_FLAGS[CLF_INCOMPLETE]
620      1607 4      THEN
621      1608 5      BEGIN
622      1609 5      KERNEL_CALL (MARK_INCOMPLETE, .CURRENT_WINDOW);
623      1610 5      ERR_EXIT (SS$$_EXBYTLM);
624      1611 4      END;
625      1612 4      IF .FCB[FCB$$_EXFCB] EQL 0 THEN EXITLOOP 0;
626      1613 4      UNTIL .WINDOW_SEGMENT[WCB$$_LINK] EQL 0
627      1614 4      DO WINDOW_SEGMENT = .WINDOW_SEGMENT[WCB$$_LINK];
628      1615 4      FCB = .FCB[FCB$$_EXFCB];
629      1616 4      HEADER = READ_HEADER (0, .FCB);
630      1617 3      END;
631      1618 3
632      1619 3      WINDOW_SEGMENT = .NEW_WINDOW[WCB$$_LINK];
633      1620 3      UNTIL .WINDOW_SEGMENT EQL 0
634      1621 3      DO
635      1622 4      BEGIN
636      1623 4      KERNEL_CALL (ADD_WINDOW, .WINDOW_SEGMENT, .PRIMARY_FCB[FCB$$_WLBL]);
637      1624 4      WINDOW_SEGMENT = .WINDOW_SEGMENT[WCB$$_LINK];
638      1625 3      END;
639      1626 3
```

```

: 640      1627 3   KERNEL_CALL (MARK_COMPLETE, .CURRENT_WINDOW);
: 641      1628 3   RETURN;
: 642      1629 3   END;
: 643      1630 3
: 644      1631 2   ! Third case; the file was never completely mapped. For this case no special
: 645      1632 2   ! precautions need to be taken. Simply loop through all the FCB's associated
: 646      1633 2   ! with the file, and create as many window segments as necessary.
: 647      1634 2   !
: 648      1635 2
: 649      1636 2   FCB = .PRIMARY_FCB;
: 650      1637 2   WINDOW_SEGMENT = .CURRENT_WINDOW;
: 651      1638 2   KERNEL_CALL (MARK_CATHEDRAL, .WINDOW_SEGMENT); !build cathedral windows
: 652      1639 2
: 653      1640 2   ! Now build the new windows using the original primary window as the base
: 654      1641 2   ! for the new window segments. This is necessary to avoid having to mung
: 655      1642 2   ! the primary window address which may reside in several places. It also
: 656      1643 2   ! means that if an error occurs, the new window created will be valid, but
: 657      1644 2   ! it will not be the same as it started out.
: 658      1645 2   !
: 659      1646 2
: 660      1647 2   UNTIL .FCB EQL 0
: 661      1648 2   DO
: 662      1649 3   BEGIN
: 663      1650 3   HEADER = READ_HEADER (0, .FCB);
: 664      1651 3   UNTIL .WINDOW_SEGMENT[WCBSL_LINK] EQL 0
: 665      1652 3   DO WINDOW_SEGMENT = .WINDOW_SEGMENT[WCBSL_LINK];
: 666      1653 3   KERNEL_CALL (TURN_WINDOW, .WINDOW_SEGMENT, .HEADER, 1, .FCB[FCBSL_STVBN]);
: 667      1654 3   IF .CLEANUP_FLAGS[CLF_INCOMPLETE]
: 668      1655 3   THEN
: 669      1656 4   BEGIN
: 670      1657 4   KERNEL_CALL (MARK_INCOMPLETE, .CURRENT_WINDOW);
: 671      1658 4   ERR_EXIT (SS$_EXBYTLM);
: 672      1659 3   END;
: 673      1660 3   FCB = .FCB[FCBSL_EXCB];
: 674      1661 2   END;
: 675      1662 2
: 676      1663 2   WINDOW_SEGMENT = .CURRENT_WINDOW[WCBSL_LINK];
: 677      1664 2   UNTIL .WINDOW_SEGMENT EQL 0
: 678      1665 2   DO
: 679      1666 3   BEGIN
: 680      1667 3   KERNEL_CALL (ADD_WINDOW, .WINDOW_SEGMENT, .PRIMARY_FCB[FCBSL_WLBL]);
: 681      1668 3   WINDOW_SEGMENT = .WINDOW_SEGMENT[WCBSL_LINK];
: 682      1669 3   END;
: 683      1670 2
: 684      1671 2   KERNEL_CALL (MARK_COMPLETE, .CURRENT_WINDOW);
: 685      1672 2   RETURN;
: 686      1673 2
: 687      1674 1   END;

```

! end of routine REMAP_FILE

```

.EXTRN DEALLOCATE, READ_HEADER
.EXTRN TURN_WINDOW, MARK_COMPLETE
.EXTRN MARK_INCOMPLETE
.EXTRN GET_MAP_POINTER
.EXTRN BUG$ _WCBFCBMNG

```

			OBFC	00000	.ENTRY	REMAP_FILE, Save R2,R3,R4,R5,R6,R7,R8,R9,-	
		5E		08 C2 00002	SUBL2	R11	1399
			0C	AA 9F 00005	PUSHAB	#8, SP	
			00	BE D5 00008	TSTL	12(BASE)	1460
				05 12 0000B	BNEQ	@0(SP)	1477
			00AC	8F BF 0000D	CHMU	1\$	
				04 00011	RET	#172	
			00	BE D0 00012 1\$:	MOVL	@0(SP), R0	1482
18	0B	50		05 E1 00016	BBC	#5, 11(R0), 3\$	
01	0B	A0		06 E1 0001B	BBC	#6, 11(R0), 2\$	1483
				04 00020	RET		
0D	0B	A0		05 E1 00021 2\$:	BBC	#5, 11(R0), 3\$	1501
08	0B	A0		06 E0 00026	BBS	#6, 11(R0), 3\$	
				50 DD 0002B	PUSHL	R0	1504
	0000V	CF		01 FB 0002D	CALLS	#1, MARK_CATHEDRAL	
				04 00032	RET		1503
			00	BE D0 00033 3\$:	MOVL	@0(SP), R0	1512
03	0B	A0		06 E0 00037	BBS	#6, 11(R0), 4\$	
			013E	31 0003C	BRW	25\$	
		52		50 D0 0003F 4\$:	MOVL	R0, WINDOW_SEGMENT	1515
		55	08	AA D0 00042	MOVL	8(BASE), FCB	1516
			20	A2 D5 00046 5\$:	TSTL	32(WINDOW_SEGMENT)	1518
				06 13 00049	BEQL	6\$	
		52	20	A2 D0 0004B	MOVL	32(WINDOW_SEGMENT), WINDOW_SEGMENT	1519
				F5 11 0004F	BRB	5\$	
	08	AE		52 D0 00051 6\$:	MOVL	WINDOW_SEGMENT, NEW_WINDOW	1520
	5B		2C	A2 D0 00055	MOVL	44(WINDOW_SEGMENT), WINDOW_ENDVBN	1522
	54		30	A2 9E 00059	MOVAB	48(R2), WINDOW_POINTER	1523
	50		16	A2 3C 0005D	MOVZWL	22(WINDOW_SEGMENT), J	1524
				50 D6 00061	J		
				09 11 00063	BRB	8\$	
	51			84 3C 00065 7\$:	MOVZWL	(WINDOW_POINTER)+, R1	1526
	5B			51 C0 00068	ADDL2	R1, WINDOW_ENDVBN	
	54			04 C0 0006B	ADDL2	#4, WINDOW_POINTER	1527
	F4			50 F5 0006E 8\$:	S0BGTR	J, 7\$	1524
	04	AE		55 D0 00071	MOVL	FCB, LAST_FCB	1531
	5B		2C	A5 D1 00075 9\$:	CMPL	44(FCB), WINDOW_ENDVBN	1534
				0A 14 00079	BGTR	10\$	
	04	AE		55 D0 0007B	MOVL	FCB, LAST_FCB	1535
	55		0C	A5 D0 0007F	MOVL	12(FCB), FCB	1536
				F0 12 00083	BNEQ	9\$	1538
	55		04	AE D0 00085 10\$:	MOVL	LAST_FCB, FCB	1539
				55 DD 00089	PUSHL	FCB	1540
				7E D4 0008B	CLRL	-(SP)	
	0000G	CF		02 FB 0008D	CALLS	#2, READ_HEADER	
	59			50 D0 00092	MOVL	R0, HEADER	
	53		2C	A5 D0 00095	MOVL	44(FCB), HEADER_VBN	1541
	50		01	A9 9A 00099	MOVZBL	1(HEADER), R0	1542
	58		6940	3E 0009D	MOVAV	(HEADER)[R0], HEADER_POINTER	
	53			5B D1 000A1	CMPL	WINDOW_ENDVBN, HEADER_VBN	1543
				73 13 000A4	BEQL	19\$	
	50		01	A9 9A 000A6 11\$:	MOVZBL	1(HEADER), R0	1544
	51		3A	A9 9A 000AA	MOVZBL	58(HEADER), R1	
	50			51 C0 000AE	ADDL2	R1, R0	
	50		6940	3E 000B1	MOVAV	(HEADER)[R0], R0	
	50			58 D1 000B5	CMPL	HEADER_POINTER, R0	

			16	1E	000B8	BGEQU	13\$		
			0000G	30	000BA	BSBW	GET MAP POINTER		1547
	53		5B	D1	000BD	CMPL	WINDOW_ENDVBN, HEADER_VBN		1548
			09	19	000C0	BLSS	12\$		
50	53		56	C1	000C2	ADDL3	HEADER_COUNT, HEADER_VBN, R0		1549
	50		5B	D1	000C6	CMPL	WINDOW_ENDVBN, R0		
			60	19	000C9	BLSS	20\$		
	53		56	C0	000CB	ADDL2	HEADER_COUNT, HEADER_VBN		1551
			D6	11	000CE	BRB	11\$		1544
	55	0C	A5	D0	000D0	MOVL	12(FCB), FCB		1553
	52	00	BE	D0	000D4	MOVL	20(SP), WINDOW_SEGMENT		1560
	51	2C	A2	D0	000D8	MOVL	44(WINDOW_SEGMENT), WINDOW_VBN		1564
	53		51	D1	000DC	CMPL	WINDOW_VBN, HEADER_VBN		1565
			27	13	000DF	BEQL	17\$		
	54	30	A2	9E	000E1	MOVAB	48(R2), WINDOW_POINTER		1566
	50	16	A2	3C	000E5	MOVZWL	22(WINDOW_SEGMENT), J		1567
			50	D6	000E9	INCL	J		
			0E	11	000EB	BRB	16\$		
	56		84	3C	000ED	MOVZWL	(WINDOW_POINTER)+, R6		1569
	51		56	C0	000F0	ADDL2	R6, WINDOW_VBN		
	54		04	C0	000F3	ADDL2	#4, WINDOW_POINTER		1570
	53		51	D1	000F6	CMPL	WINDOW_VBN, HEADER_VBN		1571
			0D	13	000F9	BEQL	17\$		
	EF		50	F5	000FB	SOBCTR	J, 15\$		1567
	52	20	A2	D0	000FE	MOVL	32(WINDOW_SEGMENT), WINDOW_SEGMENT		1573
			D4	12	00102	BNEQ	14\$		1575
			FEFF		00104	BUGW			1577
			0000*		00106	.WORD	<BUG\$ WCBFCBMNG!4>		
	55	04	AE	D0	00108	MOVL	LAST_FCB, FCB		1585
	56	20	A2	D0	0010C	MOVL	32(WINDOW_SEGMENT), NEXT_SEGMENT		1586
			52	DD	00110	PUSHL	WINDOW_SEGMENT		1587
0000V	CF		01	FB	00112	CALLS	#1, LAST_SEGMENT		
			56	D5	00117	TSTL	NEXT_SEGMENT		1588
			10	13	00119	BEQL	20\$		
	50		56	D0	0011B	MOVL	NEXT_SEGMENT, JUNK_SEGMENT		1592
	56	20	A6	D0	0011E	MOVL	32(NEXT_SEGMENT), NEXT_SEGMENT		1593
			50	DD	00122	PUSHL	JUNK_SEGMENT		1594
0000V	CF		01	FB	00124	CALLS	#1, REMOVE_WINDOW		
			EC	11	00129	BRB	18\$		1588
		2C	A5	DD	0012B	PUSHL	44(FCB)		1605
			01	DD	0012E	PUSHL	#1		
	0000G	CF	8F	BB	00130	PUSHR	#^M<R2,R9>		
7C		6A	04	FB	00134	CALLS	#4, TURN_WINDOW		
			0A	E0	00139	RBS	#10, (BASE), 29\$		1606
		0C	A5	D5	0013D	TSTL	12(FCB)		1612
			1D	13	00140	BEQL	23\$		
		20	A2	D5	00142	TSTL	32(WINDOW_SEGMENT)		1613
			06	13	00145	BEQL	22\$		
	52	20	A2	D0	00147	MOVL	32(WINDOW_SEGMENT), WINDOW_SEGMENT		1614
			F5	11	0014B	BRB	21\$		
	55	0C	A5	D0	0014D	MOVL	12(FCB), FCB		1615
			55	DD	00151	PUSHL	FCB		1616
			7E	D4	00153	CLRL	-(SP)		
	0000G	CF	02	FB	00155	CALLS	#2, READ_HEADER		
	59		50	D0	0015A	MOVL	R0, HEADER		
			CC	11	0015D	BRB	20\$		1603
50	08	AE	20	C1	0015F	ADDL3	#32, NEW_WINDOW, R0		1619

	52		60	D0	00164	MOVL	(R0), WINDOW_SEGMENT	:	
			6B	13	00167	BEQL	32\$	:	1620
	50	08	AA	D0	00169	MOVL	8(BASE), R0	:	1623
		14	A0	DD	0016D	PUSHL	20(R0)	:	
			52	DD	00170	PUSHL	WINDOW_SEGMENT	:	
0000V	CF		02	FB	00172	CALLS	#2, ADD_WINDOW	:	
	52	20	A2	D0	00177	MOVL	32(WINDOW_SEGMENT), WINDOW_SEGMENT	:	1624
			EA	11	0017B	BRB	24\$	:	1620
	55	08	AA	D0	0017D	MOVL	8(BASE), FCB	:	1636
	52	00	BE	D0	00181	MOVL	@0(SP), WINDOW_SEGMENT	:	1637
			52	DD	00185	PUSHL	WINDOW_SEGMENT	:	1638
0000V	CF		01	FB	00187	CALLS	#1, MARK_CATHEDRAL	:	
			55	D5	0018C	TSTL	FCB	:	1647
			3C	13	0018E	BEQL	31\$	:	
			55	DD	00190	PUSHL	FCB	:	1650
			7E	D4	00192	CLRL	-(SP)	:	
0000G	CF		02	FB	00194	CALLS	#2, READ_HEADER	:	
	59		50	D0	00199	MOVL	R0, HEADER	:	
		20	A2	D5	0019C	TSTL	32(WINDOW_SEGMENT)	:	1651
			06	13	0019F	BEQL	28\$	:	
	52	20	A2	D0	001A1	MOVL	32(WINDOW_SEGMENT), WINDOW_SEGMENT	:	1652
			F5	11	001A5	BRB	27\$	:	
		2C	A5	DD	001A7	PUSHL	44(FCB)	:	1653
			01	DD	001AA	PUSHL	#1	:	
		0204	8F	BB	001AC	PUSHR	#^M<R2,R9>	:	
0000G	CF		04	FB	001B0	CALLS	#4, TURN_WINDOW	:	
	6A		0A	E1	001B5	BBC	#10, (BASE), 30\$	:	1654
		00	BE	DD	001B9	PUSHL	@0(SP)	:	1657
0000G	CF		01	FB	001BC	CALLS	#1, MARK_INCOMPLETE	:	
		2A14	8F	BF	001C1	CHMU	#10772	:	1658
			04	001C5	RET			:	
	55	0C	A5	D0	001C6	MOVL	12(FCB), FCB	:	1660
			C0	11	001CA	BRB	26\$	:	1647
	50	00	BE	D0	001CC	MOVL	@0(SP), R0	:	1663
	52	20	A0	D0	001D0	MOVL	32(R0), WINDOW_SEGMENT	:	
			14	13	001D4	BEQL	33\$	:	1664
	50	08	AA	D0	001D6	MOVL	8(BASE), R0	:	1667
		14	A0	DD	001DA	PUSHL	20(R0)	:	
			52	DD	001DD	PUSHL	WINDOW_SEGMENT	:	
0000V	CF		02	FB	001DF	CALLS	#2, ADD_WINDOW	:	
	52	20	A2	D0	001E4	MOVL	32(WINDOW_SEGMENT), WINDOW_SEGMENT	:	1668
			EA	11	001E8	BRB	32\$	:	1664
		00	BE	DD	001EA	PUSHL	@0(SP)	:	1671
0000G	CF		01	FB	001ED	CALLS	#1, MARK_COMPLETE	:	
			04	001F2	RET			:	1674

; Routine Size: 499 bytes, Routine Base: \$CODE\$ + 0155

```

: 689      1675 1 GLOBAL ROUTINE LOCK_VOLUME (FUNC) : L_NORM =
: 690      1676 1
: 691      1677 1 ++
: 692      1678 1
: 693      1679 1 FUNCTIONAL DESCRIPTION:
: 694      1680 1
: 695      1681 1     This routine either locks or unlocks the current volume (or volume
: 696      1682 1     set), as indicated by the function argument. This routine must be
: 697      1683 1     called in kernel mode.
: 698      1684 1
: 699      1685 1 CALLING SEQUENCE:
: 700      1686 1     LOCK_VOLUME (ARG1)
: 701      1687 1
: 702      1688 1 INPUT PARAMETERS:
: 703      1689 1     ARG1: ACPCNTRL subfunction code
: 704      1690 1
: 705      1691 1 IMPLICIT INPUTS:
: 706      1692 1     CURRENT_VCB: VCB of volume
: 707      1693 1
: 708      1694 1 OUTPUT PARAMETERS:
: 709      1695 1     NONE
: 710      1696 1
: 711      1697 1 IMPLICIT OUTPUTS:
: 712      1698 1     NONE
: 713      1699 1
: 714      1700 1 ROUTINE VALUE:
: 715      1701 1     1
: 716      1702 1
: 717      1703 1 SIDE EFFECTS:
: 718      1704 1     lock bit in VCB either set or cleared
: 719      1705 1
: 720      1706 1 --
: 721      1707 1
: 722      1708 2 BEGIN
: 723      1709 2
: 724      1710 2 BIND_COMMON;
: 725      1711 2
: 726      1712 2 EXTERNAL ROUTINE
: 727      1713 2     ALLOCATION_LOCK : L_NORM,
: 728      1714 2     ALLOCATION_UNLOCK : L_NORM,
: 729      1715 2     DEQ_LOCK : L_NORM,      ! dequeue a lock
: 730      1716 2     TAKE_BLOCK_LOCK : L_NORM;
: 731      1717 2
: 732      1718 2 LOCAL
: 733      1719 2     RVT : REF BBLOCK;
: 734      1720 2
: 735      1721 2 IF .FUNC EQL FIB$C_LOCK_VOL
: 736      1722 2 THEN
: 737      1723 3     BEGIN
: 738      1724 3     IF .BLOCK_LOCKID EQL 0
: 739      1725 3     THEN
: 740      1726 4         BEGIN
: 741      1727 4         TAKE_BLOCK_LOCK ();
: 742      1728 4
: 743      1729 4         IF .CURRENT_VCB [VCB$W_RVN] EQL 0
: 744      1730 4         THEN
: 745      1731 4             CURRENT_VCB [VCB$V_NOALLOC] = 1
```

```

: 746      1732  4      ELSE
: 747      1733  5      BEGIN
: 748      1734  5      RVT = .CURRENT_VCB [VCB$L_RVT];
: 749      1735  5      INCR J FROM 1 TO .RVT [RVT$B_NVOLS]
: 750      1736  5      DO
: 751      1737  6          BEGIN
: 752      1738  6          LOCAL
: 753      1739  6              UCB : REF BBLOCK;
: 754      1740  6
: 755      1741  6              UCB = .VECTOR [RVT [RVT$L_UCBLST], .J-1];
: 756      1742  6
: 757      1743  6              IF .UCB NEQ 0
: 758      1744  6                  AND .BBLOCK [UCB [UCB$L_DEVCHAR], DEV$V_MNT]
: 759      1745  6              THEN
: 760      1746  6                  BBLOCK [.UCB [UCB$L_VCB], VCB$V_NOALLOC] = 1;
: 761      1747  5              END;
: 762      1748  4          END;
: 763      1749  3      END;
: 764      1750  3  ELSE
: 765      1751  2      BEGIN
: 766      1752  3      LOCAL
: 767      1753  3          FREE_SPACE;
: 768      1754  3
: 769      1755  3      IF .CURRENT_VCB [VCB$W_RVN] EQL 0
: 770      1756  3      THEN
: 771      1757  3          BEGIN
: 772      1758  4              IF .CURRENT_VCB [VCB$L_SBMAPLBN] NEQ 0
: 773      1759  4              THEN
: 774      1760  4                  CURRENT_VCB [VCB$V_NOALLOC] = 0;
: 775      1761  4              IF .BLOCK_LOCKID NEQ 0
: 776      1762  4              THEN
: 777      1763  4                  BEGIN
: 778      1764  5                      FREE_SPACE = .CURRENT_VCB [VCB$L_FREE];
: 779      1765  5                      ALLOCATION_LOCK ();
: 780      1766  5                      CURRENT_VCB [VCB$L_FREE] = .FREE_SPACE;
: 781      1767  5                      ALLOCATION_UNLOCK ();
: 782      1768  5
: 783      1769  5                      DEQ_LOCK (.BLOCK_LOCKID);
: 784      1770  5                      BLOCK_LOCKID = 0;
: 785      1771  5                  END
: 786      1772  5              END
: 787      1773  4          END
: 788      1774  3      ELSE
: 789      1775  4          BEGIN
: 790      1776  4              RVT = .CURRENT_VCB [VCB$L_RVT];
: 791      1777  4              INCR J FROM 1 TO .RVT [RVT$B_NVOLS]
: 792      1778  4              DO
: 793      1779  5                  BEGIN
: 794      1780  5                  LOCAL
: 795      1781  5                      UCB;
: 796      1782  5
: 797      1783  5                      UCB = .VECTOR [RVT [RVT$L_UCBLST], .J-1];
: 798      1784  5                      IF .UCB NEQ 0
: 799      1785  5                      THEN
: 800      1786  6                          BEGIN
: 801      1787  6                              CURRENT_UCB = .UCB;
: 802      1788  6

```

```
! end of routine LOCK_VOLUME
```

			.EXTRN	ALLOCATION_UNLOCK .EXTRN DEQ_LOCK, TAKE_BLOCK_LOCK		
		00FC	00000	.ENTRY LOCK_VOLUME, Save R2,R3,R4,R5,R6,R7	:	1675
56	FF7C	CA	9E 00002	MOVAB -132(BASE), R6	:	1708
55	98	AA	9E 00007	MOVAB -104(BASE), R5	:	
07	04	AC	D1 0000B	CMLPL FUNC, #7	:	1721
		3E	12 0000F	BNEQ 5\$	:	
		66	D5 00011	TSTL (R6)	:	1724
		37	12 00013	BNEQ 4\$	:	
0000G	CF	00	FB 00015	CALLS #0, TAKE_BLOCK_LOCK	:	1727
	50	65	D0 0001A	MOVL (R5), R0	:	1729
		OE	A0 B5 0001D	TSTW 14(R0)	:	
		06	12 00020	BNEQ 1\$	:	
OB	A0	10	88 00022	BISB2 #16, 11(R0)	:	1731
		24	11 00026	BRB 4\$	:	
52	20	A0	D0 00028 1\$:	MOVL 32(R0), RVT	:	1734
53	OB	A2	9A 0002C	MOVZBL 11(RVT), R3	:	1735
		51	D4 00030	CLRL J	:	
		14	11 00032	BRB 3\$	:	
50	40	A241	D0 00034 2\$:	MOVL 64(RVT)[J], UCB	:	1741
		OD	13 00039	BEQL 3\$	:	1743
3A	A0	03	E1 0003B	BBC #3, 58(UCB), 3\$	:	1744
	50	34	A0 D0 00040	MOVL 52(UCB), R0	:	1746

E8	0B	A0	10	88	00044	BISB2	#16, 11(R0)	:	1735
	51		53	F3	00048	AOBLEQ	R3, J, 2\$	:	1724
			0083	31	0004C	BRW	13\$	:	1756
	50		65	D0	0004F	MOVL	(R5), R0	:	
		0E	A0	B5	00052	TSTW	14(R0)	:	
			27	12	00055	BNEQ	7\$	:	
		34	A0	D5	00057	TSTL	52(R0)	:	1759
			04	13	0005A	BEQL	6\$	:	
	0B	A0	10	8A	0005C	BICB2	#16, 11(R0)	:	1761
			66	D5	00060	TSTL	(R6)	:	1762
			6E	13	00062	BEQL	13\$	:	
	50		65	D0	00064	MOVL	(R5), R0	:	1765
	54	40	A0	D0	00067	MOVL	64(R0), FREE SPACE	:	
0000G	CF		00	FB	0006B	CALLS	#0, ALLOCATION_LOCK	:	1766
	50		65	D0	00070	MOVL	(R5), R0	:	1767
40	A0		54	D0	00073	MOVL	FREE SPACE, 64(R0)	:	
0000G	CF		00	FB	00077	CALLS	#0, ALLOCATION_UNLOCK	:	1768
			4B	11	0007C	BRB	11\$	:	1770
	52	20	A0	D0	0007E	MOVL	32(R0), RVT	:	1776
	57	0B	A2	9A	00082	MOVZBL	11(RVT), R7	:	1777
			53	D4	00086	CLRL	J	:	
			37	11	00088	BRB	10\$	:	
	50	40	A243	D0	0008A	MOVL	64(RVT)[J], UCB	:	1783
			30	13	0008F	BEQL	10\$	:	1784
94	AA		50	D0	00091	MOVL	UCB, -108(BASE)	:	1788
	65	34	A0	D0	00095	MOVL	52(R0), (R5)	:	1789
	50		65	D0	00099	MOVL	(R5), R0	:	1791
		34	A0	D5	0009C	TSTL	52(R0)	:	
			04	13	0009F	BEQL	9\$	:	
	0B	A0	10	8A	000A1	BICB2	#16, 11(R0)	:	1793
			66	D5	000A5	TSTL	(R6)	:	1795
			18	13	000A7	BEQL	10\$	:	
	50		65	D0	000A9	MOVL	(R5), R0	:	1798
	54	40	A0	D0	000AC	MOVL	64(R0), FREE SPACE	:	
0000G	CF		00	FB	000B0	CALLS	#0, ALLOCATION_LOCK	:	1799
	50		65	D0	000B5	MOVL	(R5), R0	:	1800
40	A0		54	D0	000B8	MOVL	FREE SPACE, 64(R0)	:	
0000G	CF		00	FB	000BC	CALLS	#0, ALLOCATION_UNLOCK	:	1801
C5	53		57	F3	000C1	AOBLEQ	R7, J, 8\$	:	1777
			66	D5	000C5	TSTL	(R6)	:	1806
			07	13	000C7	BEQL	12\$	:	
			66	DD	000C9	PUSHL	(R6)	:	1808
0000G	CF		01	FB	000CB	CALLS	#1, DEQ_LOCK	:	
			66	D4	000D0	CLRL	(R6)	:	1810
	50		01	D0	000D2	MOVL	#1, R0	:	1816
			04	000D5	RET			:	1818

; Routine Size: 214 bytes, Routine Base: \$CODE\$ + 0348

```

: 834 1819 1 ! The remaining routines must be locked into the working set as they run
: 835 1820 1 ! at an elevated IPL.
: 836 1821 1
: 837 1822 1 LOCK_CODE;
: 838 1823 1
: 839 1824 1 ROUTINE MARK_CATHEDRAL (WINDOW) : NOVALUE =
: 840 1825 1
: 841 1826 1 !++
: 842 1827 1
: 843 1828 1 ROUTINE DESCRIPTION:
: 844 1829 1
: 845 1830 1 This routine is used to mark the specified window as a Cathedral
: 846 1831 1 window. It must be executed in kernel mode.
: 847 1832 1
: 848 1833 1 CALLING SEQUENCE:
: 849 1834 1 MARK_CATHEDRAL (ARG1)
: 850 1835 1
: 851 1836 1 INPUT PARAMETERS:
: 852 1837 1 ARG1: address of the window to mark
: 853 1838 1
: 854 1839 1 IMPLICIT INPUTS:
: 855 1840 1 none
: 856 1841 1
: 857 1842 1 OUTPUT PARAMETERS:
: 858 1843 1 none
: 859 1844 1
: 860 1845 1 IMPLICIT OUTPUTS:
: 861 1846 1 none
: 862 1847 1
: 863 1848 1 ROUTINE VALUE:
: 864 1849 1 none
: 865 1850 1
: 866 1851 1 SIDE EFFECTS:
: 867 1852 1 none
: 868 1853 1
: 869 1854 1 !--
: 870 1855 1
: 871 1856 2 BEGIN
: 872 1857 2
: 873 1858 2 MAP
: 874 1859 2 WINDOW : REF BBLOCK; ! address of the window to mark
: 875 1860 2
: 876 1861 2 LOCAL
: 877 1862 2 P : REF BBLOCK; ! copy of the window address
: 878 1863 2
: 879 1864 2 P = .WINDOW;
: 880 1865 2
: 881 1866 2 SET_IPL (IPL$_SYNCH);
: 882 1867 2
: 883 1868 2 IF NOT .P[WCB$V_COMPLETE]
: 884 1869 2 THEN
: 885 1870 3 BEGIN
: 886 1871 3 P[WCB$L_STVBN] = 1;
: 887 1872 3 P[WCB$W_NMAP] = 0;
: 888 1873 3 END;
: 889 1874 2
: 890 1875 2 P[WCB$V_CATHEDRAL] = 1; ! mark the window
```

```
: 891      1876  2
: 892      1877  2 SET_IPL (0);
: 893      1878  2
: 894      1879  2 RETURN;
: 895      1880  2
: 896      1881  1 END;
```

! end of routine MARK_CATHEDRAL

.PSECT \$LOCKEDC1\$,NOWRT,2

0000 00000 MARK_CATHEDRAL:

		50	04	AC	D0	00002	.WORD	Save nothing	: 1824
		12		08	DA	00006	MOVL	WINDOW, P	: 1864
07	0B	A0		05	E0	00009	MTPR	#8, #18	: 1866
	2C	A0		01	D0	0000E	BBS	#5, 11(P), 1\$	: 1868
			16	A0	B4	00012	MOVL	#1, 44(P)	: 1871
	0B	A0	40	8F	88	00015	CLRW	22(P)	: 1872
		12		00	DA	0001A	BISB2	#64, 11(P)	: 1875
					04	0001D	MTPR	#0, #18	: 1877
							RET		: 1881

; Routine Size: 30 bytes, Routine Base: \$LOCKEDC1\$ + 0000

```
: 898 1882 1 ROUTINE ADD_WINDOW (WINDOW, QUEUE_HEAD) : NOVALUE =
: 899 1883 1
: 900 1884 1 ++
: 901 1885 1
: 902 1886 1 FUNCTIONAL DESCRIPTION:
: 903 1887 1
: 904 1888 1 This routine adds the window specified into the queue specified. This
: 905 1889 1 routine must be called in kernel mode.
: 906 1890 1
: 907 1891 1 CALLING SEQUENCE:
: 908 1892 1 ADD_WINDOW (ARG1, ARG2)
: 909 1893 1
: 910 1894 1 INPUT PARAMETERS:
: 911 1895 1 ARG1: address of the window segment to add
: 912 1896 1 ARG2: address of the queue head
: 913 1897 1
: 914 1898 1 IMPLICIT INPUTS:
: 915 1899 1 none
: 916 1900 1
: 917 1901 1 OUTPUT PARAMETERS:
: 918 1902 1 none
: 919 1903 1
: 920 1904 1 IMPLICIT OUTPUTS:
: 921 1905 1 none
: 922 1906 1
: 923 1907 1 ROUTINE VALUE:
: 924 1908 1 none
: 925 1909 1
: 926 1910 1 SIDE EFFECTS:
: 927 1911 1 none
: 928 1912 1
: 929 1913 1 --
: 930 1914 1
: 931 1915 2 BEGIN
: 932 1916 2
: 933 1917 2 MAP
: 934 1918 2 WINDOW : REF BBLOCK, ! address of the window segment
: 935 1919 2 QUEUE_HEAD : REF BBLOCK; ! address of the queue head
: 936 1920 2
: 937 1921 2 INSQUE (.WINDOW, .QUEUE_HEAD);
: 938 1922 2
: 939 1923 2 RETURN;
: 940 1924 2
: 941 1925 1 END; ! end of routine ADD_WINDOW
```

```
0000 00000 ADD_WINDOW:
08 BC 04 BC 0E 00002 .WORD Save nothing : 1882
04 00007 INSQUE @WINDOW, @QUEUE_HEAD : 1921
RET : 1925
```

; Routine Size: 8 bytes, Routine Base: \$LOCKEDC1\$ + 001E

```

: 943      1926 1 ROUTINE REMOVE_WINDOW (WINDOW) : NOVALUE =
: 944      1927 1
: 945      1928 1 ++
: 946      1929 1
: 947      1930 1 FUNCTIONAL DESCRIPTION:
: 948      1931 1
: 949      1932 1 This routine removes the specifed window from the queue. It then
: 950      1933 1 proceeds to deallocate the window. This routine must be called in
: 951      1934 1 kernel mode.
: 952      1935 1
: 953      1936 1 CALLING SEQUENCE:
: 954      1937 1 REMOVE_WINDOW (ARG1)
: 955      1938 1
: 956      1939 1 INPUT PARAMETERS:
: 957      1940 1 ARG1: address of the window to remove
: 958      1941 1
: 959      1942 1 IMPLICIT INPUTS:
: 960      1943 1 none
: 961      1944 1
: 962      1945 1 OUTPUT PARAMETERS:
: 963      1946 1 none
: 964      1947 1
: 965      1948 1 IMPLICIT OUTPUTS:
: 966      1949 1 none
: 967      1950 1
: 968      1951 1 ROUTINE VALUE:
: 969      1952 1 none
: 970      1953 1
: 971      1954 1 SIDE EFFECTS:
: 972      1955 1 none
: 973      1956 1
: 974      1957 1 --
: 975      1958 1
: 976      1959 2 BEGIN
: 977      1960 2
: 978      1961 2 MAP
: 979      1962 2 WINDOW : REF BBLOCK; ! address of the window
: 980      1963 2
: 981      1964 2 LOCAL
: 982      1965 2 DUMMY; ! temp storage for queue entry address
: 983      1966 2
: 984      1967 2 EXTERNAL ROUTINE
: 985      1968 2 DEALLOCATE; ! deallocate system dynamic memory
: 986      1969 2
: 987      1970 2 REMQUE (.WINDOW, DUMMY);
: 988      1971 2 DEALLOCATE (.WINDOW);
: 989      1972 2
: 990      1973 2 RETURN;
: 991      1974 2
: 992      1975 1 END; ! end of routine REMOVE_WINDOW

```

```

0000 00000 REMOVE_WINDOW:
          .WORD Save nothing

```

: 1926

; Routine Size: 15 bytes, Routine Base: \$LOCKEDC1\$ + 0026

ALL
Syn
ACL
ADD
ALL
ALL
AQE
BIT
BL
BYT
CAC
CHI
CON
CUR
DAL
DAT
DEA
DIR
DYN
DYN
DYN
DYN
DYN
DYN
DYN
DYN
EXE
EXE
EXE
EXE
FCB
HEA
IND
IO
IPC
IRP
JIB
JIB
MVL
PCE
PRJ
QUC
RSM
RSM
RV1
SCH
SCH
SCH
TYF
VCE
WCE
WCE
WCE
WCE

```
: 994      1976 1 ROUTINE LAST_SEGMENT (WINDOW) : NOVALUE =
: 995      1977 1
: 996      1978 1 ++
: 997      1979 1
: 998      1980 1 FUNCTIONAL DESCRIPTION:
: 999      1981 1
: 1000     1982 1 This routine zaps the link pointer of the specified window segment
: 1001     1983 1 therefore making it the last segment in the Cathedral window.
: 1002     1984 1
: 1003     1985 1 CALLING SEQUENCE:
: 1004     1986 1 LAST_SEGMENT (ARG1)
: 1005     1987 1
: 1006     1988 1 INPUT PARAMETERS:
: 1007     1989 1 ARG1: address of the window segment
: 1008     1990 1
: 1009     1991 1 IMPLICIT INPUTS:
: 1010     1992 1 none
: 1011     1993 1
: 1012     1994 1 OUTPUT PARAMETERS:
: 1013     1995 1 none
: 1014     1996 1
: 1015     1997 1 IMPLICIT OUTPUTS:
: 1016     1998 1 none
: 1017     1999 1
: 1018     2000 1 ROUTINE VALUE:
: 1019     2001 1 none
: 1020     2002 1
: 1021     2003 1 SIDE EFFECTS:
: 1022     2004 1 none
: 1023     2005 1
: 1024     2006 1 --
: 1025     2007 1
: 1026     2008 2 BEGIN
: 1027     2009 2
: 1028     2010 2 MAP
: 1029     2011 2 WINDOW : REF BBLOCK; ! address of the window segment
: 1030     2012 2
: 1031     2013 2 WINDOW[WCB$$_LINK] = 0;
: 1032     2014 2
: 1033     2015 2 RETURN;
: 1034     2016 2
: 1035     2017 1 END; ! end of routine LAST_SEGMENT
```

0000 0000 LAST_SEGMENT:

50	04	AC	D0	00002	.WORD	Save nothing
	20	A0	D4	00006	MOVL	WINDOW, R0
			04	00009	CLRL	32(R0)
					RET	

: 1976
: 2013
: 2017

; Routine Size: 10 bytes, Routine Base: \$LOCKEDC1\$ + 0035

; 1036 2018 1

ACPCNTRL
V04-000

M 2
15-Sep-1984 23:52:53
14-Sep-1984 12:30:08

VAX-11 Bliss-32 V4.0-742
[F11X.SRC]ACPCNTRL.B32;1

Page 31
(9)

: 1037
: 1038
2019 1 END
2020 0 ELUDOM

PSECT SUMMARY

Name	Bytes	Attributes
\$CODE\$	1054 NOVEC,NOWRT, RD ;	EXE,NOSHR, LCL, REL, CON,NOPIC,ALIGN(2)
\$LOCKEDC1\$	63 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	69	0	1000	00:01.8

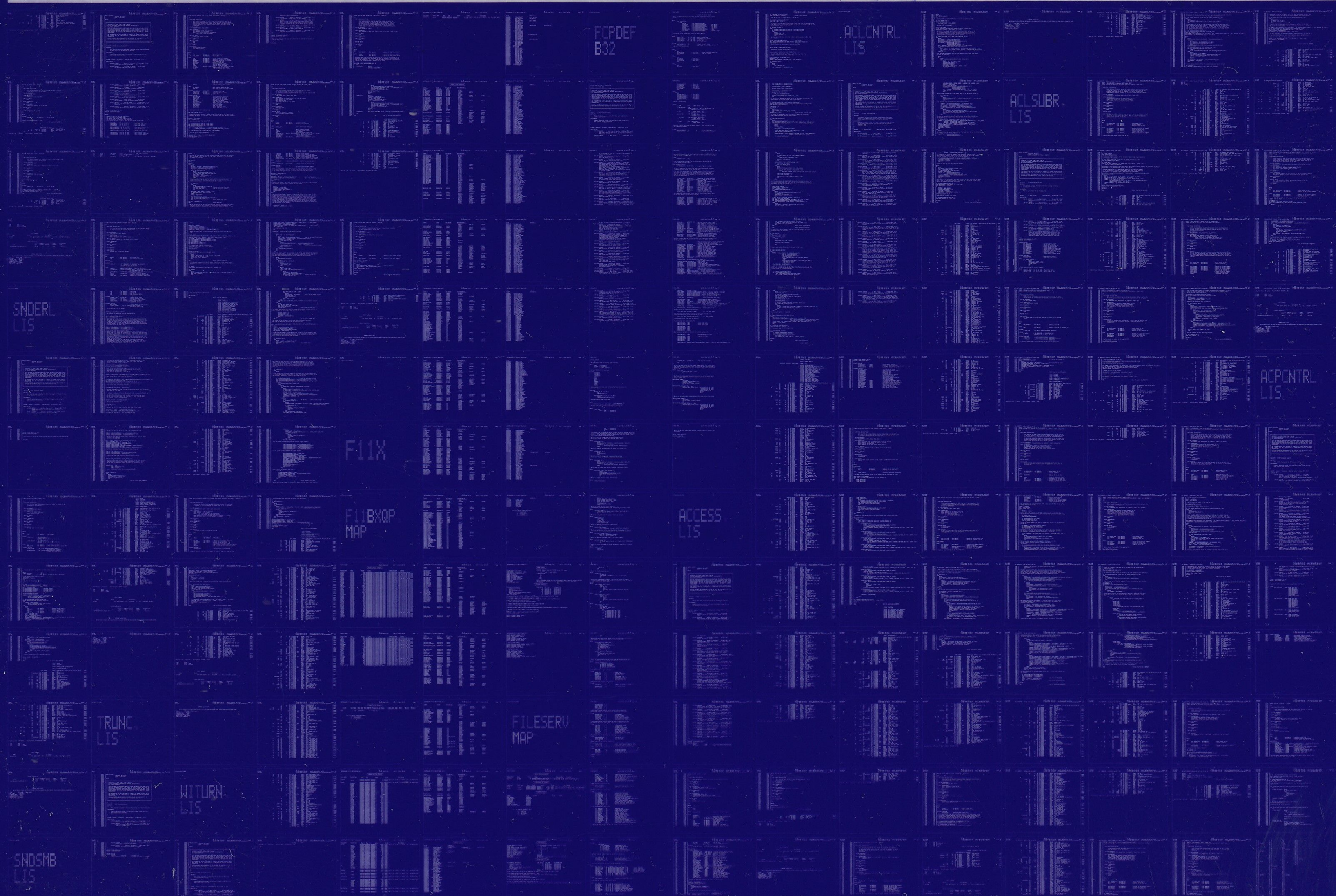
COMMAND QUALIFIERS

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

: Size: 1117 code + 0 data bytes
: Run Time: 00:58.4
: Elapsed Time: 02:12.8
: Lines/CPU Min: 2076
: Lexemes/CPU-Min: 50879
: Memory Used: 327 pages
: Compilation Complete

0167 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY



0168

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY