



```

SSSSSSSS MM MM AAAAAA LL
SSSSSSSS MM MM AAAAAA LL
SS MMMM MMMM AA AA LL
SS MMMM MMMM AA AA LL
SS MM MM MM AA AA LL
SSSSSS MM MM AA AA LL
SSSSSS MM MM AA AA LL
SS MM MM AAAAAAAAAA LL
SS MM MM AAAAAAAAAA LL
SS MM MM AA AA LL
SS MM MM AA AA LL
SSSSSSSS MM MM AA AA LLLLLLLLLL
SSSSSSSS MM MM AA AA LLLLLLLLLL
000000 000000 CCCCCCCC
00 00 00 CC
00 00 00 CC
00 00 00 CC
00 00 00 CC
00 00 00 CC
00 00 00 CC
00 00 00 CC
00 00 00 CC
00 00 00 CC
00 00 00 CC
000000 000000 CCCCCCCC
000000 000000 CCCCCCCC
.....
.....
.....
.....

```

```

LL I I I I I I SSSSSSSS
LL I I I I I I SSSSSSSS
LL I I SS
LL I I SS
LL I I SS
LL I I SSSSSS
LL I I SSSSSS
LL I I SS
LL I I SS
LL I I SS
LL I I SS
LLLLLLLLLLLL I I I I I I SSSSSSSS
LLLLLLLLLLLL I I I I I I SSSSSSSS

```

SM  
VO

:

```

: 1 0001 0 MODULE SMALOC (
: 2 0002 0 LANGUAGE (BLISS32),
:001 :CDS0005 0003 0 IDENT = 'V04-001'
: 4-1 0004 0 ) =
: 5 0005 1 BEGIN
: 6 0006 1
: 7 0007 1
: 8 0008 1
: 9 0009 1
:10 0010 1 *
:11 0011 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
:12 0012 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
:13 0013 1 * ALL RIGHTS RESERVED.
:14 0014 1 *
:15 0015 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
:16 0016 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
:17 0017 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
:18 0018 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
:19 0019 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
:20 0020 1 * TRANSFERRED.
:21 0021 1 *
:22 0022 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
:23 0023 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
:24 0024 1 * CORPORATION.
:25 0025 1 *
:26 0026 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
:27 0027 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
: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 contains the routines that manipulate the volume
:38 0038 1 storage bitmap. These include the routines to allocate a contiguous
:39 0039 1 area, deallocate an area, and the basic bitmap scanner.
:40 0040 1 Also included are the routines that manage the extent cache.
:41 0041 1
:42 0042 1 ENVIRONMENT:
:43 0043 1
:44 0044 1 STARLET operating system, including privileged system services
:45 0045 1 and internal exec routines.
:46 0046 1
:47 0047 1 --
:48 0048 1
:49 0049 1
:50 0050 1 AUTHOR: Andrew C. Goldstein, CREATION DATE: 21-Feb-1977 18:42
:51 0051 1
:52 0052 1 MODIFIED BY:
:53 0053 1
:001 :CDS0005 0054 1 V04-001 CDS0005 Christian D. Saether 15-Nov-1984
:002 :CDS0005 0055 1 Expand test for clusterness to look at clu$gl_club.
:003 :CDS0005 0056 1
: 54 0057 1 V03-012 ACG0445 Andrew C. Goldstein, 21-Aug-1984 20:48

```



55	0058	1	Fix handling of null extent cache in RETURN_BLOCKS
56	0059	1	
57	0060	1	V03-011 ACG0438 Andrew C. Goldstein, 1-Aug-1984 18:51
58	0061	1	Add extent cache interlock logic; remove kernel calls,
59	0062	1	fold in UPDATE_FREE and SET_SMVBN routines. Use central
60	0063	1	dequeue routine.
61	0064	1	
62	0065	1	V03-010 LMP0257 L. Mark Pilant, 25-Jun-1984 9:42
63	0066	1	Use double precision when calculation cluster round-up to
64	0067	1	insure that the cluster calculation is unsigned.
65	0068	1	
66	0069	1	V03-009 CDS0004 Christian D. Saether 29-Dec-1983
67	0070	1	Use L_NORM linkage and BIND_COMMON macro.
68	0071	1	
69	0072	1	V03-008 CDS0003 Christian D. Saether 25-Sep-1983
70	0073	1	Manually merge in STJ3106.
71	0074	1	
72	0075	1	V03-007 STJ3106 Steven T. Jeffreys, 20-Jun-1983
73	0076	1	- Implement Erase On Extend (EOE).
74	0077	1	
75	0078	1	V03-006 CDS0002 Christian D. Saether 13-Sep-1983
76	0079	1	Change interface to allocation serialization routine.
77	0080	1	
78	0081	1	V03-005 CDS0001 Christian D. Saether 13-May-1983
79	0082	1	Serialize storage allocation/deallocation activity.
80	0083	1	
81	0084	1	V03-004 STJ3081 Steven T. Jeffreys, 30-Mar-1983
82	0085	1	- Added CHANNEL parameter to ERASE_BLOCKS call.
83	0086	1	
84	0087	1	V03-003 STJ3062 Steven T. Jeffreys, 18-Mar-1982
85	0088	1	- Added call to ERASE_BLOCKS from RETURN_BLOCKS.
86	0089	1	- Added ERASE_REQUESTED parameter to RETURN_BLOCKS.
87	0090	1	
88	0091	1	V03-002 ACG0298 Andrew C. Goldstein, 25-Aug-1982 16:32
89	0092	1	Detect attempts to create negative extent cache entries
90	0093	1	
91	0094	1	V03-001 ACG45949 Andrew C. Goldstein, 8-Jun-1982 16:11
92	0095	1	Prevent volume free space from going negative
93	0096	1	
94	0097	1	V02-014 ACG43131 Andrew C. Goldstein, 4-Jan-1982 18:11
95	0098	1	Fix spurious allocation failures in approx. placed allocation
96	0099	1	
97	0100	1	V02-013 ACG0229 Andrew C. Goldstein, 23-Dec-1981 22:10
98	0101	1	Count extent cache hits and misses
99	0102	1	
100	0103	1	V02-012 ACG38789 Andrew C. Goldstein, 1-Jul-1981 19:48
101	0104	1	Check for running out bit count in cylinder round up
102	0105	1	
103	0106	1	V02-011 ACG0195 Andrew C. Goldstein, 3-Mar-1981 22:54
104	0107	1	Fix 4096 block boundary problem by checking zero in BITSCAN
105	0108	1	
106	0109	1	V02-010 ACG0180 Andrew C. Goldstein, 10-Sep-1980 14:44
107	0110	1	Fix cluster and cylinder rounding in extent cache allocator
108	0111	1	
109	0112	1	V02-009 ACG0172 Andrew C. Goldstein, 9-May-1980 10:42
110	0113	1	Check map pointer count for non-zero in RETURN_BLOCKS
111	0114	1	



```

: 112      0115 1 | V02-008 ACG0167 Andrew C. Goldstein, 16-Apr-1980 19:28
: 113      0116 1 | Previous revision history moved to F11B.REV
: 114      0117 1 | **
: 115      0118 1 |
: 116      0119 1 |
: 117      0120 1 | LIBRARY 'SYSS$LIBRARY:LIB.L32';
: 118      0121 1 | REQUIRE 'SRCS:FCPDEF.B32';
: 119      1112 1 |
: 120      1113 1 |
: 121      1114 1 |
: 122      1115 1 | Modes of operation of the bit scanner.
: 123      1116 1 |
: 124      1117 1 |
: 125      1118 1 | LITERAL
: 126      1119 1 | FIND_SET = 0, | find first one
: 127      1120 1 | FIND_CLEAR = 1, | find first zero
: 128      1121 1 | SET_BITS = 2, | set n bits
: 129      1122 1 | CLEAR_BITS = 3; | clear n bits
: 130      1123 1 |
: 131      1124 1 |
: 132      1125 1 | FORWARD ROUTINE
: 133      1126 1 | ALLOC_BLOCKS : L_NORM,
: 134      1127 1 | RETURN_BLOCKS : L_NORM NOVALUE,
: 135      1128 1 | INIT_EXT_CACHE : L_NORM NOVALUE, ! set up extent cache lock
: 136      1129 1 | ALLOC_EXTENT : L_NORM, | allocate entry from extent cache
: 137      1130 1 | RETURN_EXTENT : L_NORM, | return entry to extent cache
: 138      1131 1 | PURGE_EXTENT : L_NORM NOVALUE, ! return cache entries back to bitmap
: 139      1132 1 | REMOVE_EXTENT : L_NORM, | remove entry from extent cache
: 140      1133 1 | ALLOC_BITMAP : L_NORM, | allocate blocks from storage bitmap
: 141      1134 1 | RETURN_BITMAP : L_NORM NOVALUE, ! return blocks to storage bitmap
: 142      1135 1 | BITSCAN : L_NORM;

```

```

144 117 1 GLOBAL ROUTINE ALLOC_BLOCKS (FIB, BLOCKS_NEEDED, START_LBN, BLOCKS_ALLOC) : L_NORM =
145 1137 1
146 1138 1
147 1139 1
148 1140 1
149 1141 1
150 1142 1
151 1143 1
152 1144 1
153 1145 1
154 1146 1
155 1147 1
156 1148 1
157 1149 1
158 1150 1
159 1151 1
160 1152 1
161 1153 1
162 1154 1
163 1155 1
164 1156 1
165 1157 1
166 1158 1
167 1159 1
168 1160 1
169 1161 1
170 1162 1
171 1163 1
172 1164 1
173 1165 1
174 1166 1
175 1167 1
176 1168 1
177 1169 1
178 1170 1
179 1171 1
180 1172 1
181 1173 1
182 1174 1
183 1175 1
184 1176 2
185 1177 2
186 1178 2
187 1179 2
188 1180 2
189 1181 2
190 1182 2
191 1183 2
192 1184 2
193 1185 2
194 1186 2
195 1187 2
196 1188 2
197 1189 2
198 1190 2
199 1191 2
200 1192 2

GLOBAL ROUTINE ALLOC_BLOCKS (FIB, BLOCKS_NEEDED, START_LBN, BLOCKS_ALLOC) : L_NORM =
++
FUNCTIONAL DESCRIPTION:
    This routine allocates a single contiguous area of disk. It first
    attempts allocation from the extent cache. If that fails, it performs
    the allocation from the storage bitmap.

    As part of system security, the blocks allocated will be erased
    before returning the extent to the caller.

CALLING SEQUENCE:
    ALLOC_BLOCKS (ARG1, ARG2, ARG3, ARG4)

INPUT PARAMETERS:
    ARG1: address of FIB for this operation
    ARG2: number of blocks to allocate

IMPLICIT INPUTS:
    CURRENT_VCB: VCB of volume
    CURRENT_UCB: UCB of volume

OUTPUT PARAMETERS:
    ARG3: address of longword to store starting LBN
    ARG4: address of longword to store block count

IMPLICIT OUTPUTS:
    LOC_LBN: placement LBN of allocation or 0

ROUTINE VALUE:
    1 if successful allocation
    0 if failure

SIDE EFFECTS:
    storage map, VCB, and extent cache modified

--
BEGIN
MAP
    FIB          : REF BBLOCK;    ! FIB of operation

LITERAL
    ALLOC_RETRIES = 3;           ! Number of times to retry allocation

LOCAL
    ERASED      : status of erase operation
    ATTEMPTS    : number of attempts at cache allocation
    STATUS      : status return value
    CACHE       : REF BBLOCK;    ! pointer to main cache block
    EXTENT_CACHE : REF BBLOCK;    ! pointer to extent cache
    TEMP        : VECTOR [2];    ! quadword temp for EMUL & EDIV
    EXT_LIMIT   : local longword copy of extent limit parameter
    DUMMY       : dummy to receive remainder from EDIV

```



```

201      1193      2      CACHE_TOTAL,      ! total disk space to allocate into cache
202      1194      2      LBN.      ! LBN being allocated
203      1195      2      COUNT;      ! block count being allocated
204      1196
205      1197      BIND
206      1198      DUMMY_FIB      = UPLIT (REP FIB% EXTDATA OF (BYTE (0)));
207      1199      ! default FIB for allocation for cache
208      1200
209      1201      BIND_COMMON;
210      1202
211      1203      EXTERNAL ROUTINE
212      1204      ALLOCATION_LOCK : L_NORM,      ! serialize allocation/deallocation
213      1205      ERASE_BLOCKS : L_NORM,      ! Erase blocks before reusing them
214      1206      ALLOCATION_UNLOCK : L_NORM NOVALUE, ! release allocation lock.
215      1207      RELEASE_LOCKBASIS : L_NORM,      ! release buffers under specified lock
216      1208      DEQ_LOCK : L_NORM,      ! dequeue a lock
217      1209      CACHE_LOCK : L_NORM;      ! acquire cache sync lock
218      1210
219      1211      EXTERNAL
001 : CDS0005 212      1212      CLUSGL_CLUB : ADDRESSING_MODE (GENERAL),
220      1213      PMSSGL_EXTHIT : ADDRESSING_MODE (GENERAL),
221      1214      ! count of extent cache hits
222      1215      PMSSGL_EXTMISS : ADDRESSING_MODE (GENERAL);
223      1216      ! count of extent cache misses
224      1217
225      1218      ! Serialize processing against other storage/header allocation/deallocation.
226      1219      !
227      1220
228      1221      ALLOCATION_LOCK ();
229      1222
230      1223      ! First attempt to allocate the space from the extent cache. Note that
231      1224      ! a placed allocation can actually split a cache entry; therefore, if the
232      1225      ! cache is full after the allocation, purge it to half.
233      1226
234      1227      CACHE = .CURRENT_VCB[VCBSL_CACHE];
235      1228      EXTENT_CACHE = .CACHE[VCAS_EXTCACHE];
236      1229      IF (STATUS = ALLOC_EXTENT (.FIB, .BLOCKS_NEEDED, .START_LBN, .BLOCKS_ALLOC))
237      1230      THEN
238      1231      BEGIN
239      1232      IF .EXTENT_CACHE[VCASW_EXTCOUNT] GEQU .EXTENT_CACHE[VCASW_EXTSIZE]
240      1233      THEN
241      1234      BEGIN
242      1235      PMSSGL_EXTMISS = .PMSSGL_EXTMISS + 1;
243      1236      PURGE_EXTENT (.EXTENT_CACHE[VCASW_EXTSIZE] / 2, -1);
244      1237      END
245      1238      ELSE
246      1239      PMSSGL_EXTHIT = .PMSSGL_EXTHIT + 1;
247      1240      END
248      1241
249      1242      ! If the cache allocation failed, attempt allocation from the bitmap.
250      1243      ! If this fails, purge the cache if there is anything in it, to make
251      1244      ! the bitmap consistent. Then attempt allocation from the bitmap again.
252      1245      !
253      1246
254      1247      ELSE
255      1248      BEGIN
256      1249      PMSSGL_EXTMISS = .PMSSGL_EXTMISS + 1;

```



```

: 257      1250      3
: 258      1251      3
: 259      1252      3
: 260      1253      4
: 261      1254      5
: 262      1255      4
: 263      1256      4
: 264      1257      4 ! Can't get the space from the bitmap as is. Purge back the extent cache,
: 265      1258      4 ! and, if we're in a cluster, ask for a flush of all others and try
: 266      1259      4 ! once more.
: 267      1260      4
: 268      1261      4
: 269      1262      4
: 270      1263      4
: 001 ! CDS0005 1264      4
: 271      1265      4
: 272      1266      5
: 273      1267      5
: 274      1268      5
: 275      1269      5
: 276      1270      5
: 277      1271      5
: 278      1272      5
: 279      1273      5
: 280      1274      5
: 281      1275      4
: 282      1276      3
: 283      1277      3
: 284      1278      3 ! If extent caching is not shut off, now refill the cache from the
: 285      1279      3 ! bitmap block currently in memory.
: 286      1280      3
: 287      1281      3
: 288      1282      3
: 289      1283      3
: 290      1284      3
: 291      1285      3
: 292      1286      3
: 293      1287      4
: 294      1288      4
: 295      1289      4
: 296      1290      4
: 297      1291      4
: 298      1292      4
: 299      1293      4
: 300      1294      5
: 301      1295      5
: 302      1296      5
: 303      1297      5
: 304      1298      5
: 305      1299      5
: 306      1300      5
: 307      1301      4
: 308      1302      3
: 309      1303      2
: 310      1304      2
: 311      1305      2 ! If we successfully allocated something, erase the space if called for
: 312      1306      2 ! and deduct it from the volume's free space.

```



```

: 313      1307      2      :
: 314      1308      2      :
: 315      1309      2      IF .STATUS
: 316      1310      2      THEN
: 317      1311      2      BEGIN
: 318      1312      2      IF NOT .CURRENT_VCB[VCBSV_NOHIGHWATER]
: 319      1313      2      THEN ERASE_BLOCKS (..START_LBN, ..BLOCKS_ALLOC, ..IO_CHANNEL);
: 320      1314      2      CURRENT_VCB[VCBSL_FREE] = ..CURRENT_VCB[VCBSL_FREE] = ..BLOCKS_ALLOC;
: 321      1315      2      IF .CURRENT_VCB[VCBSL_FREE] LSS 0
: 322      1316      2      THEN CURRENT_VCB[VCBSL_FREE] = 0;
: 323      1317      2      END;
: 324      1318      2
: 325      1319      2      RETURN .STATUS;
: 326      1320      2
: 327      1321      1      END;

```

! end of routine ALLOC\_BLOCKS

		.TITLE	SMALOC
		.IDENT	\V04-001\
		.PSECT	\$CODE\$,NOWRT,2
00	00000	P.AAA:	.BYTE 0
00	00001		.BYTE 0
00	00002		.BYTE 0
00	00003		.BYTE 0
00	00004		.BYTE 0
00	00005		.BYTE 0
00	00006		.BYTE 0
00	00007		.BYTE 0
00	00008		.BYTE 0
00	00009		.BYTE 0
00	0000A		.BYTE 0
00	0000B		.BYTE 0
00	0000C		.BYTE 0
00	0000D		.BYTE 0
00	0000E		.BYTE 0
00	0000F		.BYTE 0
00	00010		.BYTE 0
00	00011		.BYTE 0
00	00012		.BYTE 0
00	00013		.BYTE 0
00	00014		.BYTE 0
00	00015		.BYTE 0
00	00016		.BYTE 0
00	00017		.BYTE 0
00	00018		.BYTE 0
00	00019		.BYTE 0
00	0001A		.BYTE 0
00	0001B		.BYTE 0
00	0001C		.BYTE 0
00	0001D		.BYTE 0
00	0001E		.BYTE 0
00	0001F		.BYTE 0

DUMMY\_FIB= P.AAA  
.EXTRN ALLOCATION\_LOCK

					.EXTRN ERASE_BLOCKS, ALLOCATION_UNLOCK	
					.EXTRN RELEASE_LOCKBASIS	
					.EXTRN DEQ_LOCK, CACHE_LOCK	
					.EXTRN CLUSGL_CLUB, PMSSGL_EXTHIT	
					.EXTRN PMSSGL_EXTMISS	
					.ENTRY ALLOC_BLOCKS, Save R2,R3,R4,R5,R6,R7	1136
				57 C0000000G 00 00FC 00000	MOVAB PMSSGL_EXTMISS, R7	
				5E 14 C2 00009	SUBL2 #20, SP	
				56 98 AA 9E 0000C	MOVAB -104(BASE), R6	1198
	0000G			CF 00 FB 00010	CALLS #0, ALLOCATION_LOCK	1221
				50 66 D0 00015	MOVL (R6), R0	1227
				53 58 A0 D0 00018	MOVL 88(R0), CACHE	
				52 04 A3 D0 0001C	MOVL 4(CACHE), EXTENT_CACHE	1228
				7E 0C AC 7D 00020	MOVQ START_LBN, -(SP)	1229
				7E 04 AC 7D 00024	MOVQ FIB, -(SP)	
	0000V			CF 04 04 FB 00028	CALLS #4, ALLOC_EXTENT	
				55 50 D0 0002D	MOVL R0, STATUS	
				22 55 E9 00030	BLBC STATUS, 3\$	
				62 02 A2 B1 00033	CMPW 2(EXTENT_CACHE), (EXTENT_CACHE)	1232
				13 1F 00037	BLSSU 1\$	
				67 D6 00039	INCL PMSSGL_EXTMISS	1235
				7E 01 CE 0003B	MNEGL #1, -(SP)	1236
				50 62 3C 0003E	MOVZWL (EXTENT_CACHE), R0	
	7E			50 02 C7 00041	DIVL3 #2, R0, -(SP)	
	0000V			CF 02 FB 00045	CALLS #2, PURGE_EXTENT	
				06 11 0004A	BRB 2\$	1232
				00000000G 00 D6 0004C	INCL PMSSGL_EXTHIT	1239
				00CA 31 00052	BRW 9\$	1229
				67 D6 00055	INCL PMSSGL_EXTMISS	1249
				54 02 D0 00057	MOVL #2, J	1251
				7E 7E D4 0005A	CLRL -(SP)	1254
				7E 0C AC 7D 0005C	MOVQ START_LBN, -(SP)	
				7E 04 AC 7D 00060	MOVQ FIB, -(SP)	
	0000V			CF 05 FB 00064	CALLS #5, ALLOC_BITMAP	
				55 50 D0 00069	MOVL R0, STATUS	
				4F 55 E8 0006C	BLPS STATUS, 6\$	
				7E 7C 0006F	CLRQ -(SP)	1262
	0000V			CF 02 FB 00071	CALLS #2, PURGE_EXTENT	
				50 94 AA D0 00076	MOVL -108(BASE), R0	1263
				3D 3C A0 E9 0007A	BLBC 60(R0), 5\$	
				00000000G 00 D5 0007E	TSTL CLUSGL_CLUB	1264
				35 13 00084	BEQL 5\$	
				7E 01 CE 00086	MNEGL #1, -(SP)	1268
	0000G			CF 01 FB 00089	CALLS #1, RELEASE_LOCKBASIS	
	0000G			CF 00 FB 0008E	CALLS #0, ALLOCATION_UNLOCK	1269
				50 66 D0 00093	MOVL (R6), R0	1270
				50 0E A0 3C 00096	MOVZWL 14(R0), R0	
	50			50 18 78 0009A	ASHL #24, R0, R0	
				50 02 88 0009E	BISB2 #2, BIT_FILE_ID	
				6E D4 000A1	CLRL LOCK_ID	1271
				01 DD 000A3	PUSHL #1	1272
				04 AE 9F 000A5	PUSHAB LOCK_ID	
				50 DD 000A8	PUSHL BIT_FILE_ID	
	0000G			CF 03 FB 000AA	CALLS #3, CACHE_LOCK	
	0000G			CF 00 FB 000AF	CALLS #0, ALLOCATION_LOCK	1273
				6E DD 000B4	PUSHL LOCK_ID	1274



		0000G	CF		01	FB	000B6		CALLS	#1, DEQ_LOCK	
			9C		54	F5	000BB	5\$:	SOBGTR	J, 4\$	1251
	OC	OB	A3		01	E0	000BE	6\$:	BBS	#1, 11(CACHE), 7\$	1282
		0000V	CF		53	DD	000C3		PUSHL	CACHE	1283
	50	OB	A3		01	FB	000C5		CALLS	#1, INIT_EXT_CACHE	
					01	E1	000CA		BBC	#1, 11(CACHE), 9\$	1285
					AA	D4	000CF	7\$:	CLRL	32(BASE)	1288
			51	20	A2	3C	000D2		MOVZWL	8(EXTENT_CACHE), EXT_LIMIT	1289
			50	08	66	D0	000D6		MOVL	(R6), R0	1290
OC	AE	00	A0		51	7A	000D9		EMUL	EXT_LIMIT, 64(R0), #0, TEMP	
	50	53	OC	000003E8	8F	7B	000E0		EDIV	#1000, TEMP, CACHE_TOTAL, DUMMY	1291
			50		62	3C	000EA	8\$:	MOVZWL	(EXTENT_CACHE), R0	1292
			50		02	C6	000ED		DIVL2	#2, R0	
	50	02	A2		00	ED	000F0		CMPZV	#0, #16, 2(EXTENT_CACHE), R0	
			10		27	1E	000F6		BGEQU	9\$	
					01	DD	000F8		PUSHL	#1	1295
				08	AE	9F	000FA		PUSHAB	COUNT	
				10	AE	9F	000FD		PUSHAB	LBN	
					53	DD	00100		PUSHL	CACHE_TOTAL	
		0000V	CF	FEDA	05	FB	00102		PUSHAB	DUMMY_FIB	
			11		50	E9	0010B		CALLS	#5, ALOC_BITMAP	
					04	AE	DD	0010E	BLBC	R0, 9\$	
				04	AE	DD	00111		PUSHL	COUNT	1297
		0000V	CF		02	FB	00114		PUSHL	LBN	
			53	04	AE	C2	00119		CALLS	#2, RETURN_EXTENT	
					CB	14	0011D		SUBL2	COUNT, CACHE_TOTAL	1298
			2A		55	E9	0011F	9\$:	BGTR	8\$	1299
			50		66	D0	00122		BLBC	STATUS, 11\$	1309
			A0		04	E0	00125		MOVL	(R6), R0	1312
	OF	53	A0		CA	DD	0012A		BBS	#4, 83(R0), 10\$	
				FF78	BC	DD	0012E		PUSHL	-136(BASE)	1313
				10	BC	DD	00131		PUSHL	@BLOCKS_ALLOC	
		0000G	CF	OC	03	FB	00134		PUSHL	@START_LBN	
			50		66	D0	00139	10\$:	CALLS	#3, ERASE_BLOCKS	
		40	A0	10	BC	C2	0013C		MOVL	(R6), R0	1314
			50		66	D0	00141		SUBL2	@BLOCKS_ALLOC, 64(R0)	
				40	A0	D5	00144		MOVL	(R6), R0	1315
					03	18	00147		TSTL	64(R0)	
				40	A0	D4	00149		BGEQ	11\$	
			50		55	D0	0014C	11\$:	CLRL	64(R0)	1316
					04	0014F			MOVL	STATUS, R0	1319
									RET		1321

; Routine Size: 336 bytes, Routine Base: \$CODE\$ + 0020

```

329 1322 1 GLOBAL ROUTINE RETURN_BLOCKS (START_LBN, BLOCK_COUNT, ERASE_REQUESTED) : L_NORM NOVALUE =
330 1323 1
331 1324 1 !++
332 1325 1
333 1326 1 FUNCTIONAL DESCRIPTION:
334 1327 1
335 1328 1 This routine returns a single contiguous area to the storage pool.
336 1329 1 If there is space in the cache, the blocks are simply returned to
337 1330 1 the cache. If the cache is full, it first purges some of the cache
338 1331 1 entries and then returns the blocks.
339 1332 1
340 1333 1 CALLING SEQUENCE:
341 1334 1 RETURN_BLOCKS (ARG1, ARG2, ARG3)
342 1335 1
343 1336 1 INPUT PARAMETERS:
344 1337 1 ARG1: starting LBN to free
345 1338 1 ARG2: number of blocks to free
346 1339 1 ARG3: boolean. 1 if blocks are to be erased, 0 if not.
347 1340 1
348 1341 1 IMPLICIT INPUTS:
349 1342 1 CURRENT_VCB: VCB of volume
350 1343 1 CURRENT_UCB: UCB of device
351 1344 1
352 1345 1 OUTPUT PARAMETERS:
353 1346 1 NONE
354 1347 1
355 1348 1 IMPLICIT OUTPUTS:
356 1349 1 NONE
357 1350 1
358 1351 1 ROUTINE VALUE:
359 1352 1 NONE
360 1353 1
361 1354 1 SIDE EFFECTS:
362 1355 1 storage map, VCB, and extent cache modified
363 1356 1
364 1357 1 --
365 1358 1
366 1359 2 BEGIN
367 1360 2
368 1361 2 LOCAL
369 1362 2 STATUS, : local storage for routine status
370 1363 2 CACHE : REF BBLOCK, : pointer to main cache block
371 1364 2 EXTENT_CACHE : REF BBLOCK, : pointer to extent cache
372 1365 2 TEMP : VECTOR [2], : quadword temp for EMUL & EDIV
373 1366 2 EXT_LIMIT, : local longword copy of extent limit parameter
374 1367 2 DUMMY, : dummy to receive remainder from EDIV
375 1368 2 CACHE_LIMIT; : total disk space to allocate into cache
376 1369 2
377 1370 2 BIND_COMMON;
378 1371 2
379 1372 2 EXTERNAL
380 1373 2 PMS$GL_EXTHIT : ADDRESSING_MODE (GENERAL),
381 1374 2 : count of extent cache hits
382 1375 2 PMS$GL_EXTMISS : ADDRESSING_MODE (GENERAL);
383 1376 2 : count of extent cache misses
384 1377 2
385 1378 2 EXTERNAL ROUTINE

```



```
386 1379          ALLOCATION_LOCK : L_NORM,  
387 1380          ERASE_BLOCKS   : L_NORM;          ! Erase blocks before reusing them  
388 1381  
389 1382  
390 1383          ! First check the block count for non-zero.  
391 1384          !  
392 1385  
393 1386          IF .BLOCK_COUNT EQL 0  
394 1387          THEN ERR_EXIT (SS$_BADFILEHDR);  
395 1388  
396 1389          ! Check the blocks being returned against the volume size.  
397 1390          !  
398 1391  
399 1392          IF .START_LBN + .BLOCK_COUNT GTRU .CURRENT_UCB[UCB$_L_MAXBLOCK]  
400 1393          THEN ERR_EXIT (SS$_BADFILEHDR);  
401 1394  
402 1395          ! Check that the start LBN and count are integral multiples of the  
403 1396          ! cluster factor. If not, reject the operation on grounds of a bad  
404 1397          ! file header.  
405 1398          !  
406 1399  
407 1400          IF .START_LBN MOD .CURRENT_VCB[VCB$_W_CLUSTER] NEQ 0  
408 1401          OR .BLOCK_COUNT MOD .CURRENT_VCB[VCB$_W_CLUSTER] NEQ 0  
409 1402          THEN ERR_EXIT (SS$_BADFILEHDR);  
410 1403  
411 1404          ! Before returning the blocks, erase them if need be.  
412 1405          ! Notify the user if an error is encountered.  
413 1406          !  
414 1407  
415 1408          IF .ERASE_REQUESTED  
416 1409          THEN  
417 1410              IF NOT (STATUS = ERASE_BLOCKS (.START_LBN, .BLOCK_COUNT, .IO_CHANNEL))  
418 1411              THEN  
419 1412                  ERR_STATUS (.STATUS);  
420 1413  
421 1414          ! Serialize processing against other storage/header allocation/deallocation.  
422 1415          !  
423 1416  
424 1417          ALLOCATION_LOCK();  
425 1418  
426 1419          ! Attempt to activate the extent cache if it is not active. If it refuses  
427 1420          ! to activate (e.g., is null, or is inhibited due to interlocks), return  
428 1421          ! the space directly to the bitmap.  
429 1422          !  
430 1423  
431 1424          CACHE = .CURRENT_VCB[VCB$_L_CACHE];  
432 1425          EXTENT_CACHE = .CACHE[VCAS$_EXTCACHE];  
433 1426  
434 1427          IF NOT .CACHE[VCAS$_EXTC_VALID]  
435 1428          THEN INIT_EXT_CACHE (.CACHE);  
436 1429  
437 1430          IF NOT .CACHE[VCAS$_EXTC_VALID]  
438 1431          THEN  
439 1432              BEGIN  
440 1433                  RETURN_BITMAP (.START_LBN, .BLOCK_COUNT);  
441 1434                  PMSSGL_EXTMISS = .PMSSGL_EXTMISS + 1;  
442 1435              END
```

```

: 443 1436
: 444 1437
: 445 1438
: 446 1439
: 447 1440
: 448 1441
: 449 1442
: 450 1443
: 451 1444
: 452 1445
: 453 1446
: 454 1447
: 455 1448
: 456 1449
: 457 1450
: 458 1451
: 459 1452
: 460 1453
: 461 1454
: 462 1455
: 463 1456
: 464 1457
: 465 1458
: 466 1459
: 467 1460
: 468 1461
: 469 1462

! Return the blocks to the cache. If the cache is full or if it now contains
! more space than we want, then purge it to half and/or below the limit.

ELSE
BEGIN
IF NOT RETURN_EXTENT (.START_LBN, .BLOCK_COUNT)
THEN ERR_EXIT (SS$_BADFILEHDR);

EXT_LIMIT = .EXTENT_CACHE[V$CASW_EXTLIMIT];
EMUL (EXT_LIMIT, CURRENT_VCB[V$CB$SL_FREE], %REF (0), TEMP);
EDIV (%REF (1000), TEMP, %CACHE_LIMIT, DUMMY);
IF .EXTENT_CACHE[V$CASW_EXTCOUNT] GEQU .EXTENT_CACHE[V$CASW_EXTSIZE]
OR .EXTENT_CACHE[V$CASL_EXTTOTAL] GTRU .CACHE_LIMIT
THEN
BEGIN
PURGE_EXTENT (.EXTENT_CACHE[V$CASW_EXTSIZE] / 2, .CACHE_LIMIT);
PM$SGL_EXTMISS = .PM$SGL_EXTMISS + 1;
END
ELSE
PM$SGL_EXTHIT = .PM$SGL_EXTHIT + 1;
END;

CURRENT_VCB[V$CB$SL_FREE] = .CURRENT_VCB[V$CB$SL_FREE] + .BLOCK_COUNT;

END;
! end of routine RETURN_BLOCKS

```

				000C 00000	.ENTRY	RETURN_BLOCKS, Save R2,R3	: 1322
	5E		08	C2 00002	SUBL2	#8, SP	
			08	AC D5 00005	TSTL	BLOCK_COUNT	: 1386
			03	12 00008	BNEQ	1\$	
			0090	31 0000A	BRW	4\$	
	51	04	08	AC C1 0000D	ADDL3	BLOCK_COUNT, START_LBN, R1	: 1392
			94	AA D0 00013	MOVL	-108(BASE), R0	
		00B0		51 D1 00017	CMPL	R1, 176(R0)	
				7F 1A 00C1C	BGTRU	4\$	
			50	98 AA D0 0001E	MOVL	-104(BASE), R0	: 1400
			51	3C A0 3C 00022	MOVZWL	60(R0), R1	
7E	00	04		01 7A 00026	EMUL	#1, START_LBN, #0, -(SP)	
51	51	8E		51 7B 0002C	EDIV	R1, (SP)+, R1, R1	
				51 D5 00031	TSTL	R1	
				68 12 00033	BNEQ	4\$	
			50	3C AC 3C 00035	MOVZWL	60(R0), R0	: 1401
				01 7A 00039	EMUL	#1, BLOCK_COUNT, #0, -(SP)	
7E	00	08		50 7B 0003F	EDIV	R0, (SP)+, R0, R0	
50	50	8E		50 D5 00044	TSTL	R0	
				55 12 00046	BNEQ	4\$	
		18		0C AC E9 00048	BLBC	ERASE_REQUESTED, 2\$	: 1408
				FF78 CA DD 0004C	PUSHL	-136(BASE)	: 1410
		7E		04 AC 7D 00050	MOVQ	START_LBN, -(SP)	
		0000G		03 FB 00054	CALLS	#3, ERASE_BLOCKS	
		08		50 EB 00059	BLBS	STATUS, 2\$	



		04	80	AA	E9	0005C		BLBC	-128(BASE), 2\$	:	1412
	80	AA		50	B0	00060		MOVW	STATUS, -128(BASE)	:	
	0000G	CF		00	FB	00064	2\$:	CALLS	#0, ALLOCATION_LOCK	:	1417
		50	98	AA	D0	00069		MOVL	-104(BASE), R0	:	1424
		52	58	A0	D0	0006D		MOVL	88(R0), CACHE	:	
		53	04	A2	D0	00071		MOVL	4(CACHE), EXTENT_CACHE	:	1425
17	0B	A2		01	E0	00075		BBS	#1, 11(CACHE), 3\$	:	1427
				52	DD	0007A		PUSHL	CACHE	:	1428
	0000V	CF		01	FB	0007C		CALLS	#1, INIT_EXT_CACHE	:	
0B	0B	A2		01	E0	00081		BBS	#1, 11(CACHE), 3\$	:	1430
		7E	04	AC	7D	00086		MOVQ	START_LBN, -(SP)	:	1433
	0000V	CF		02	FB	0008A		CALLS	#2, RETURN_BITMAP	:	
				42	11	0008F		BRB	7\$	:	1434
		7E	04	AC	7D	00091	3\$:	MOVQ	START_LBN, -(SP)	:	1443
	0000V	CF		02	FB	00095		CALLS	#2, RETURN_EXTENT	:	
		05		50	E8	0009A		BLBS	R0, 5\$	:	
			0810	8F	BF	0009D	4\$:	CHMU	#2064	:	1444
				04	00	00A1		RET		:	
		51	08	A3	3C	000A2	5\$:	MOVZWL	8(EXTENT_CACHE), EXT_LIMIT	:	1446
		50	98	AA	D0	000A6		MOVL	-104(BASE), R0	:	1447
6E	00	A0	40	51	7A	000AA		EMUL	EXT_LIMIT, 64(R0), #0, TEMP	:	
51	50	6E	000003E8	8F	7B	000B0		EDIV	#1000, TEMP, CACHE_LIMIT, DUMMY	:	1448
		63	02	A3	B1	000B9		CMPW	2(EXTENT_CACHE), (EXTENT_CACHE)	:	1449
				06	1E	000BD		BGEQU	6\$	:	
		50	04	A3	D1	000BF		CMP	4(EXTENT_CACHE), CACHE_LIMIT	:	1450
				16	1B	000C3		BLEQU	8\$	:	
				50	DD	000C5	6\$:	PUSHL	CACHE_LIMIT	:	1453
		50		63	3C	000C7		MOVZWL	(EXTENT_CACHE), R0	:	
7E		50		02	C7	000CA		DIVL3	#2, R0, -(SP)	:	
	0000V	CF		02	FB	000CE		CALLS	#2, PURGE_EXTENT	:	
			00000000G	00	D6	000D3	7\$:	INCL	PM\$GL_EXTMISS	:	1454
				06	11	000D9		BRB	9\$	:	1449
			00000000G	00	D6	000DB	8\$:	INCL	PM\$GL_EXTHIT	:	1457
		50		00	D0	000E1	9\$:	MOVL	-104(BASE), R0	:	1460
	40	A0	08	AC	C0	000E5		ADDL2	BLOCK_COUNT, 64(R0)	:	
				04	00	00EA		RET		:	1462

: Routine Size: 235 bytes, Routine Base: \$CODE\$ + 0170

```

: 471 1463 1 GLOBAL ROUTINE INIT_EXT_CACHE (CACHE) : L_NORM NOVALUE =
: 472 1464 1
: 473 1465 1 !++
: 474 1466 1
: 475 1467 1 FUNCTIONAL DESCRIPTION:
: 476 1468 1
: 477 1469 1 This routine sets up the extent cache interlock as necessary
: 478 1470 1 and marks the cache valid, if this is possible, considering
: 479 1471 1 dismount state of the volume and write access to the storage map.
: 480 1472 1
: 481 1473 1 CALLING SEQUENCE:
: 482 1474 1 INIT_EXT_CACHE (CACHE)
: 483 1475 1
: 484 1476 1 INPUT PARAMETERS:
: 485 1477 1 CACHE: pointer to main cache block
: 486 1478 1
: 487 1479 1 IMPLICIT INPUTS:
: 488 1480 1 NONE
: 489 1481 1
: 490 1482 1 OUTPUT PARAMETERS:
: 491 1483 1 NONE
: 492 1484 1
: 493 1485 1 IMPLICIT OUTPUTS:
: 494 1486 1 NONE
: 495 1487 1
: 496 1488 1 ROUTINE VALUE:
: 497 1489 1 NONE
: 498 1490 1
: 499 1491 1 SIDE EFFECTS:
: 500 1492 1 cache marked valid, lock taken out
: 501 1493 1
: 502 1494 1 --
: 503 1495 1
: 504 1496 2 BEGIN
: 505 1497 2
: 506 1498 2 MAP
: 507 1499 2 CACHE : REF BBLOCK; ! pointer to cache block
: 508 1500 2
: 509 1501 2 LOCAL
: 510 1502 2 EXT_CACHE : REF BBLOCK, ! pointer to file ID cache
: 511 1503 2 BITMAP_FID; ! lock basis for index file
: 512 1504 2
: 513 1505 2 BIND_COMMON;
: 514 1506 2
: 001 :CDS0005 1507 2 EXTERNAL
: 002 :CDS0005 1508 2 CLUSGL_CLUB : ADDRESSING_MODE (GENERAL);
: 003 :CDS0005 1509 2
: 515 1510 2 EXTERNAL ROUTINE
: 516 1511 2 CACHE_LOCK : L_NORM; ! acquire special cache lock
: 517 1512 2
: 518 1513 2
: 519 1514 2 ! If the cache is not currently marked valid, attempt to take out the
: 520 1515 2 ! cache lock if we are in a cluster and may do so.
: 521 1516 2
: 522 1517 2
: 523 1518 2 EXT_CACHE = .CACHE[VCASL_EXTCACHE];
: 524 1519 2 IF NOT .BBLOCK [CURRENT_OCB[UCBSL_DEVCHAR], DEV$V_DMT]

```





```

: 542 1537 1 ROUTINE ALLOC_EXTENT (FIB, BLOCKS_NEEDED, START_LBN, BLOCKS_ALLOC) : L_NORM =
: 543 1538 1
: 544 1539 1 ++
: 545 1540 1
: 546 1541 1 FUNCTIONAL DESCRIPTION:
: 547 1542 1
: 548 1543 1 This routine allocates a single contiguous area of disk from
: 549 1544 1 the extent cache. Mode of allocation is determined by the
: 550 1545 1 allocation control in the FIB.
: 551 1546 1
: 552 1547 1 CALLING SEQUENCE:
: 553 1548 1 ALLOC_EXTENT (ARG1, ARG2, ARG3, ARG4)
: 554 1549 1
: 555 1550 1 INPUT PARAMETERS:
: 556 1551 1 ARG1: address of FIB for this operation
: 557 1552 1 ARG2: number of blocks to allocate
: 558 1553 1
: 559 1554 1 IMPLICIT INPUTS:
: 560 1555 1 CURRENT_VCB: ADDRESS OF VCB IN PROCESS
: 561 1556 1 CURRENT_UCB: ADDRESS OF UCB IN PROCESS
: 562 1557 1
: 563 1558 1 OUTPUT PARAMETERS:
: 564 1559 1 ARG3: address of longword to store starting LBN
: 565 1560 1 ARG4: address of longword to store block count
: 566 1561 1
: 567 1562 1 IMPLICIT OUTPUTS:
: 568 1563 1 LOC_LBN: placement LBN of allocation or 0
: 569 1564 1 NONE
: 570 1565 1
: 571 1566 1 ROUTINE VALUE:
: 572 1567 1 1 if successful allocation
: 573 1568 1 0 if failure
: 574 1569 1
: 575 1570 1 SIDE EFFECTS:
: 576 1571 1 Extent cache modified
: 577 1572 1
: 578 1573 1 --
: 579 1574 1
: 580 1575 2 BEGIN
: 581 1576 2
: 582 1577 2 MAP
: 583 1578 2 FIB : REF BBLOCK; ! FIB or operation
: 584 1579 2
: 585 1580 2 LABEL
: 586 1581 2 CACHE_SEARCH; ! extent cache search procedure
: 587 1582 2
: 588 1583 2 REGISTER
: 589 1584 2 EXTENT_LIST : REF BBLOCKVECTOR [,8]; ! pointer to extent list
: 590 1585 2
: 591 1586 2 LOCAL
: 592 1587 2 EXTENT_CACHE : REF BBLOCK, ! pointer to extent cache
: 593 1588 2 BLOCK_COUNT, ! blocks needed rounded up to cluster
: 594 1589 2 J, ! loop and extent list index
: 595 1590 2 LBN, ! LBN of current extent
: 596 1591 2 COUNT, ! block count of current extent
: 597 1592 2 CYL_SIZE, ! size in blocks of volume's cylinder
: 598 1593 2 CYL_BOUNDARY; ! LBN of next cylinder boundary

```



```
599 1594 2
600 1595 2 BIND_COMMON;
601 1596 2
602 1597 2 ! Search the extent cache. If placement is specified, check for a match
603 1598 2 ! against the placement LBN.
604 1599 2
605 1600 2
606 1601 2 CACHE_SEARCH: BEGIN
607 1602 2
608 1603 2 BLOCK_COUNT = ((.BLOCKS_NEEDED+.CURRENT_VCB[VCBSW_CLUSTER]-1)
609 1604 2 / .CURRENT_VCB[VCBSW_CLUSTER]) * .CURRENT_VCB[VCBSW_CLUSTER];
610 1605 2 EXTENT_CACHE = .BLOCK [.CURRENT_VCB[VCBSL_CACHE], VCASL_EXTCACHE];
611 1606 2 EXTENT_LIST = EXTENT_CACHE[VCASQ_EXTLIST];
612 1607 2
613 1608 2 J = 1;
614 1609 2 WHILE .J LEQU .EXTENT_CACHE[VCASW_EXTCOUNT]
615 1610 2 DO
616 1611 4 BEGIN
617 1612 4 LBN = .EXTENT_LIST[J-1, VCASL_EXTLBN];
618 1613 4 COUNT = .EXTENT_LIST[J-1, VCASL_EXTBLOCKS];
619 1614 4
620 1615 4 IF .LOC_LBN EQL 0
621 1616 5 OR (.LOC_LBN GEQU .LBN AND .LOC_LBN LSSU .LBN + .COUNT)
622 1617 4 THEN
623 1618 5 BEGIN
624 1619 5
625 1620 5 ! If placement is specified, adjust the base LBN and count accordingly.
626 1621 5 ! Likewise, if on-cylinder allocation is requested, move the LBN to the
627 1622 5 ! cylinder boundary. Then adjust to the cluster boundary.
628 1623 5
629 1624 5
630 1625 5 IF .LOC_LBN NEQ 0 THEN LBN = .LOC_LBN / .CURRENT_VCB[VCBSW_CLUSTER]
631 1626 5 * .CURRENT_VCB[VCBSW_CLUSTER];
632 1627 5 IF .FIB[FIBSV_ONCYL]
633 1628 5 THEN
634 1629 6 BEGIN
635 1630 6 CYL_SIZE = .CURRENT_UCB[UCBSB_SECTORS]
636 1631 6 * .CURRENT_UCB[UCBSB_TRACKS]
637 1632 6 / .CURRENT_VCB[VCBSB_BLOCKFACT];
638 1633 6 CYL_BOUNDARY = (.LBN / .CYL_SIZE + 1) * .CYL_SIZE;
639 1634 6 IF .CYL_BOUNDARY - .LBN LSSU .BLOCKS_NEEDED
640 1635 6 THEN
641 1636 7 BEGIN
642 1637 7 IF NOT .FIB[FIBSV_EXACT]
643 1638 9 THEN LBN = ((.CYL_BOUNDARY + .CURRENT_VCB[VCBSW_CLUSTER] - 1)
644 1639 7 / .CURRENT_VCB[VCBSW_CLUSTER]) * .CURRENT_VCB[VCBSW_CLUSTER]
645 1640 7 ELSE RETURN 0;
646 1641 6 END;
647 1642 5 END;
648 1643 5
649 1644 5 IF .LBN GEQU .EXTENT_LIST[J-1, VCASL_EXTLBN] + .COUNT
650 1645 5 THEN COUNT = 0
651 1646 5 ELSE COUNT = .COUNT + .EXTENT_LIST[J-1, VCASL_EXTLBN] - .LBN;
652 1647 5
653 1648 5 ! If the size is sufficient at this point, we win. If not, and the allocation
654 1649 5 ! is neither exact nor on-cylinder, try backing off the adjustments made
655 1650 5 ! above. Then check the size again; if the allocation is non-contiguous
```



```

: 656      1651 5 ! or if the size is big enough, this is it.
: 657      1652 5 !
: 658      1653 5
: 659      1654 5     IF .COUNT GEQU .BLOCK_COUNT
: 660      1655 5     THEN LEAVE CACHE_SEARCH;
: 661      1656 5
: 662      1657 5     IF .LOC_LBN NEQ 0
: 663      1658 5     AND NOT .FIB[FIB$V_ONCYL]
: 664      1659 5     AND NOT .FIB[FIB$V_EXACT]
: 665      1660 5     THEN
: 666      1661 6         BEGIN
: 667      1662 6             COUNT = MINU (.BLOCK_COUNT, .EXTENT_LIST[J-1, VCASL_EXTBLOCKS]);
: 668      1663 6             LBN = .EXTENT_LIST[J-1, VCASL_EXTLBN]
: 669      1664 6                 + .EXTENT_LIST[J-1, VCASL_EXTBLOCKS]
: 670      1665 6                 - .COUNT;
: 671      1666 5         END;
: 672      1667 5     IF .COUNT GEQU .BLOCK_COUNT
: 673      1668 6     OR (.COUNT NEQ 0
: 674      1669 6         AND NOT .FIB[FIB$V_ALCON]
: 675      1670 6         AND NOT .FIB[FIB$V_ALCONB])
: 676      1671 5     THEN LEAVE CACHE_SEARCH;
: 677      1672 4     END;
: 678      1673 4     J = .J + 1;
: 679      1674 3     END;                                     ! end of cache search loop
: 680      1675 3
: 681      1676 3     RETURN 0;                                 ! whole cache searched - nothing found
: 682      1677 3
: 683      1678 2     END;                                     ! end of block CACHE_SEARCH
: 684      1679 2
: 685      1680 2 ! We get here if we find a suitable cache entry. Deduct the count needed
: 686      1681 2 ! from the count in the entry. If the result is zero, squish out the entry.
: 687      1682 2 !
: 688      1683 2 !
: 689      1684 2 !
: 690      1685 2     COUNT = MINU (.COUNT, .BLOCK_COUNT);
: 691      1686 2     IF .COUNT EQL 0
: 692      1687 2     THEN BUG_CHECK (MAPCNTZER, FATAL, 'found zero extent in cache');
: 693      1688 2
: 694      1689 2     EXTENT_LIST[J-1, VCASL_EXTBLOCKS] = .EXTENT_LIST[J-1, VCASL_EXTBLOCKS] - .COUNT;
: 695      1690 2     IF .EXTENT_LIST[J-1, VCASL_EXTBLOCKS] EQL 0
: 696      1691 2     THEN
: 697      1692 3         BEGIN
: 698      1693 3             CHSMOVE ((.EXTENT_CACHE[VCASW_EXTCOUNT]-.J)*8,
: 699      1694 3                 EXTENT_LIST[J, VCASL_EXTBLOCKS],
: 700      1695 3                 EXTENT_LIST[J-1, VCASL_EXTBLOCKS]);
: 701      1696 3             EXTENT_CACHE[VCASW_EXTCOUNT] = .EXTENT_CACHE[VCASW_EXTCOUNT] - 1;
: 702      1697 3         END
: 703      1698 3
: 704      1699 3 ! Otherwise the allocation is only part of the extent. If it is from the
: 705      1700 3 ! front of the extent, recompute the starting LBN of the extent.
: 706      1701 3 !
: 707      1702 3
: 708      1703 2     ELSE IF .EXTENT_LIST[J-1, VCASL_EXTLBN] EQL .LBN
: 709      1704 2     THEN
: 710      1705 3         EXTENT_LIST[J-1, VCASL_EXTLBN] = .EXTENT_LIST[J-1, VCASL_EXTLBN] + .COUNT
: 711      1706 3
: 712      1707 2 ! If the allocation is from the end of the extent, no further action is necessary.
```



```

: 713      1708 2  ! If it is from the middle, we must split the extent. To do so, shuffle the
: 714      1709 2  ! remainder of the extent list up by one, bump the entry count, and compute
: 715      1710 2  ! the split entries.
: 716      1711 2  !
: 717      1712 2  !
: 718      1713 2  ELSE IF .EXTENT_LIST[J-1, VCASL_EXTLBN] + .EXTENT_LIST[J-1, VCASL_EXTBLOCKS] NEQ .LBN
: 719      1714 2  THEN
: 720      1715 2  BEGIN
: 721      1716 2  CHSMOVE ((.EXTENT_CACHE[VCASW_EXTCOUNT]-J)*8,
: 722      1717 2  EXTENT_LIST[J, VCASL_EXTBLOCKS],
: 723      1718 2  EXTENT_LIST[J+1, VCASL_EXTBLOCKS]);
: 724      1719 2  EXTENT_CACHE[VCASW_EXTCOUNT] = .EXTENT_CACHE[VCASW_EXTCOUNT] + 1;
: 725      1720 2  EXTENT_LIST[J, VCASL_EXTLBN] = .COUNT + .LBN;
: 726      1721 2  EXTENT_LIST[J, VCASL_EXTBLOCKS] = .EXTENT_LIST[J-1, VCASL_EXTBLOCKS]
: 727      1722 2  + .EXTENT_LIST[J-1, VCASL_EXTLBN]
: 728      1723 2  - .LBN;
: 729      1724 2  EXTENT_LIST[J-1, VCASL_EXTBLOCKS] = .EXTENT_LIST[J-1, VCASL_EXTBLOCKS]
: 730      1725 2  - .EXTENT_LIST[J, VCASL_EXTBLOCKS];
: 731      1726 2  END;
: 732      1727 2  !
: 733      1728 2  .START_LBN = .LBN;
: 734      1729 2  .BLOCKS_ALLOC = .COUNT;
: 735      1730 2  EXTENT_CACHE[VCASL_EXTTOTAL] = .EXTENT_CACHE[VCASL_EXTTOTAL] - .COUNT;
: 736      1731 2  !
: 737      1732 2  RETURN 1;
: 738      1733 2  !
: 739      1734 1  END;
! end of routine ALLOC_EXTENT

```

.EXTRN BUG\$\_MAPCNTZER

		OBFC 0000 ALLOC_EXTENT:				
				.WORD	Save R2,R3,R4,R5,R6,R7,R8,R9,R11	: 1537
				SUBL2	#12, SP	
08	5E	98	AA 9E 00005	MOVAB	-104(BASE), 8(SP)	: 1593
	50	08	BE D0 0000A	MOVL	@8(SP), R0	: 1603
	51	3C	A0 3C 0000E	MOVZWL	60(R0), R1	
	51	08	AC C0 00012	ADDL2	BLOCKS_NEEDED, R1	
			51 D7 00016	DECL	R1	
	52	3C	A0 3C 00018	MOVZWL	60(R0), R2	: 1604
	51		52 C6 0001C	DIVL2	R2, R1	
	53	3C	A0 3C 0001F	MOVZWL	60(R0), R3	
	51		53 C4 00023	MULL2	R3, BLOCK_COUNT	
	50	08	BE D0 00026	MOVL	@8(SP), R0	: 1605
	50	58	A0 D0 0002A	MOVL	88(R0), R0	
	57	04	A0 D0 0002E	MOVL	4(R0), EXTENT_CACHE	
	56	2C	A7 9E 00032	MOVAB	44(R7), EXTENT_LIST	: 1606
	5B		01 D0 00036	MOVL	#1, J	: 1608
5B	02	A7	10 00 ED 00039 1\$:	CMPZV	#0, #16, 2(EXTENT_CACHE), J	: 1609
			03 1E 0003F	BGEQU	3\$	
			018D 31 00041 2\$:	BRW	20\$	
	54		664B 7E 00044 3\$:	MOVAQ	(EXTENT_LIST)[J], R4	: 1612
	59	FC	A4 D0 00048	MOVL	-4(R4), LBN	
	52		54 D0 0004C	MOVL	R4, R2	: 1613
	58	F8	A2 D0 0004F	MOVL	-8(R2), COUNT	
	53	20	AA D0 00053	MOVL	32(BASE), R3	: 1615

				11	13	00057		BEQL	6\$			
		59		53	D1	00059		CMPL	R3, LBN			1616
				03	1E	0005C		BGEQU	5\$			
				00D0	31	0005E	4\$:	BRW	13\$			
	52			58	C1	00061	5\$:	ADDL3	COUNT, LBN, R2			
		59		53	D1	00065		CMPL	R3, R2			
		52		F4	1E	00068		BGEQU	4\$			
				53	D5	0006A	6\$:	TSTL	R3			1625
				12	13	0006C		BEQL	7\$			
		52	08	BE	D0	0006E		MOVL	@8(SP), R2			
		55	3C	A2	3C	00072		MOVZWL	60(R2), R5			
		53		55	C6	00076		DIVL2	R5, R3			
		59	3C	A2	3C	00079		MOVZWL	60(R2), LBN			1626
		59		53	C4	0007D		MULL2	R3, LBN			
		53	04	AC	D0	00080	7\$:	MOVL	FIB, R3			1627
	4C		20	A3	01	00084		BBC	#1, 32(R3), 8\$			
		52		94	AA	00089		MOVL	-108(BASE), R2			1630
		55		44	A2	0008D		MOVZBL	68(R2), R5			1631
		52		45	A2	00091		MOVZBL	69(R2), R2			
		52		55	C4	00095		MULL2	R5, R2			
		55	08	BE	D0	00098		MOVL	@8(SP), R5			1632
		6E	52	A5	9A	0009C		MOVZBL	82(R5), (SP)			
04	AE			6E	C7	000A0		DIVL3	(SP), R2, CYL_SIZE			
	52			52	AE	000A5	04	DIVL3	CYL_SIZE, LBN, R2			1633
				52	D6	000AA		INCL	R2			
	50			52	AE	000AC	04	MULL3	CYL_SIZE, R2, CYL_BOUNDARY			
	52		08	50	C3	000B1		SUBL3	LBN, CYL_BOUNDARY, R2			1634
				AC	D1	000B5		CMPL	R2, BLOCKS_NEEDED			
				82	1A	000B9		BGEQU	8\$			
				52	A3	000BB	20	BLBS	32(R3), 2\$			1637
				52	BE	000BF	08	MOVL	@8(SP), R2			1638
				52	3C	000C3		ADDL2	#60, R2			
				52	62	000C6		MOVZWL	(R2), R2			
				55	FF	000C9	A240	MOVAB	-1(R2)[CYL_BOUNDARY], R5			
				55	52	000CE		DIVL2	R2, R5			1639
	59			55	52	000D1		MULL3	R2, R5, LBN			
	52			58	FC	000D5	8\$:	ADDL3	-4(R4), COUNT, R2			1644
				52	59	000DA		CMPL	LBN, R2			
					04	000DD		BLSSU	9\$			
					58	000DF		CLRL	COUNT			1645
					09	000E1		BRB	10\$			
	52			58	FC	000E3	9\$:	ADDL3	-4(R4), COUNT, R2			1646
	58			52	59	000E8		SUBL3	LBN, R2, COUNT			
				51	58	000EC	10\$:	CMPL	COUNT, BLOCK_COUNT			1654
					45	000EF		BGEQU	14\$			
					20	000F1		TSTL	32(BASE)			1657
					29	000F4		BEQL	12\$			
	24	20	A3	01	E0	000F6		BBS	#1, 32(R3), 12\$			1658
				20	A3	000FB		BLBS	32(R3), 12\$			1659
					54	000FF		MOVL	R4, R2			1662
					55	00102		MOVL	BLOCK_COUNT, R5			
			F8	A2	55	00105		CMPL	R5, -8(R2)			
					04	00109		BLEQU	11\$			
					55	0010B	F8	MOVL	-8(R2), R5			
					58	0010F		MOVL	R5, COUNT			
					52	00112		MOVL	R4, R2			1664
54	FC	A4	F8	A2	C1	00115		ADDL3	-8(R2), -4(R4), R4			



59		54	58	C3	0011B		SUBL3	COUNT, R4, LBN	1665
		51	58	D1	0011F	12\$:	CMP	COUNT, BLOCK_COUNT	1667
			12	1E	00122		BGEQU	14\$	
			58	D5	00124		TSTL	COUNT	1668
			09	13	00126		BEQL	13\$	
		05	A3	E8	00128	16	BLBS	22(R3), 13\$	1669
05	16	A3	01	E1	0012C		BBC	#1, 22(R3), 14\$	1670
			5B	D6	00131	13\$:	INCL	J	1673
			FF03	31	00133		BRW	1\$	1609
		50	58	D0	00136	14\$:	MOVL	COUNT, R0	1685
		51	50	D1	00139		CMP	R0, BLOCK_COUNT	
			03	1B	0013C		BLEQU	15\$	
		50	51	D0	0013E		MOVL	BLOCK_COUNT, R0	
		58	50	D0	00141	15\$:	MOVL	R0, COUNT	
			04	12	00144		BNEQ	16\$	1686
				FEFF	00146		BUGW		1687
				0000*	00148		.WORD	<BUG\$ MAPCNTZER!4>	
			F8 A64B	7F	0014A	16\$:	PUSHAQ	-8(EXTENT_LIST)[J]	1689
		9E	58	C2	0014E		SUBL2	COUNT, a(SP)+	
		50	F8 A64B	7E	00151		MOVAQ	-8(EXTENT_LIST)[J], R0	1690
			60	D5	00156		TSTL	(R0)	
			16	12	00158		BNEQ	17\$	
		51	02	A7	3C	0015A	MOVZWL	2(EXTENT_CACHE), R1	1693
		51		5B	C2	0015E	SUBL2	J, R1	
		51		08	C4	00161	MULL2	#8, R1	
			664B	7F	00164		PUSHAQ	(EXTENT_LIST)[J]	1695
60		9E	51	28	00167		MOVC3	R1, a(SP)+, (R0)	
			02	A7	B7	0016B	DECW	2(EXTENT_CACHE)	1696
			51	11	0016E		BRB	19\$	1690
		59	04	A0	D1	00170	17\$:	CMP	4(R0), LBN
				06	12	00174		BNEQ	18\$
		04	A0	58	C0	00176	ADDL2	COUNT, 4(R0)	1705
				45	11	0017A	BRB	19\$	
50		04	A0	60	C1	0017C	18\$:	ADDL3	(R0), 4(R0), R0
			59	50	D1	00181		CMP	R0, LBN
				3B	13	00184		BEQL	19\$
		50	02	A7	3C	00186	MOVZWL	2(EXTENT_CACHE), R0	1716
		50		5B	C2	0018A	SUBL2	J, R0	
		50		08	C4	0018D	MULL2	#8, R0	
			08	A64B	7F	00190	PUSHAQ	8(EXTENT_LIST)[J]	1718
			664B	7F	00194		PUSHAQ	(EXTENT_LIST)[J]	
9E		9E	50	28	00197		MOVC3	R0, a(SP)+, a(SP)+	
			02	A7	B6	0019B	INCW	2(EXTENT_CACHE)	1719
			04	A64B	7F	0019E	PUSHAQ	4(EXTENT_LIST)[J]	1720
9E		58	59	C1	001A2		ADDL3	LBN, COUNT, a(SP)+	
		50	F8 A64B	7E	001A6		MOVAQ	-8(EXTENT_LIST)[J], R0	1721
50		60	04	A0	C1	001AB	ADDL3	4(R0), (R0), R0	1722
			664B	7F	001B0		PUSHAQ	(EXTENT_LIST)[J]	1723
9E		50	59	C3	001B3		SUBL3	LBN, R0, a(SP)+	
			F8 A64B	7F	001B7		PUSHAQ	-8(EXTENT_LIST)[J]	1725
			664B	7F	001BB		PUSHAQ	(EXTENT_LIST)[J]	
		9E	9E	C2	001BE		SUBL2	a(SP)+, -a(SP)+	
	0C	BC	59	D0	001C1	19\$:	MOVL	LBN, aSTART LBN	1728
	10	BC	58	D0	001C5		MOVL	COUNT, aBLOCKS_ALLOC	1729
	04	A7	58	C2	001C9		SUBL2	COUNT, 4(EXTENT_CACHE)	1730
		50	01	D0	001CD		MOVL	#1, R0	1732
			04	001D0			RET		

SMALOC  
V04-001

G 8  
8-Jan-1985 18:35:14  
2-Oct-1984 12:43:38

VAX-11 Bliss-32 V4.0-742  
[F11X.BUGSRC]SMALOC.B32;1

Page 22  
(5)

50 D4 001D1 20\$: CLRL R0  
04 001D3 RET

: 1734  
:

: Routine Size: 468 bytes, Routine Base: \$CODE\$ + 02B4



```

: 741 1735 1 ROUTINE RETURN_EXTENT (START_LBN, BLOCK_COUNT) : L_NORM =
: 742 1736 1
: 743 1737 1 |++
: 744 1738 1 |
: 745 1739 1 | FUNCTIONAL DESCRIPTION:
: 746 1740 1 |
: 747 1741 1 |     This routine returns the indicated extent to the extent cache.
: 748 1742 1 |     It searches the cache to insert the entry in LBN order, and merges
: 749 1743 1 |     it with any adjacent entries. If the extent overlaps existing
: 750 1744 1 |     entries, an error return is made.
: 751 1745 1 |
: 752 1746 1 |
: 753 1747 1 | CALLING SEQUENCE:
: 754 1748 1 |     RETURN_EXTENT (ARG1, ARG2)
: 755 1749 1 |
: 756 1750 1 | INPUT PARAMETERS:
: 757 1751 1 |     ARG1: starting LBN of extent
: 758 1752 1 |     ARG2: block count
: 759 1753 1 |
: 760 1754 1 | IMPLICIT INPUTS:
: 761 1755 1 |     CURRENT_VCB: VCB of volume
: 762 1756 1 |
: 763 1757 1 | OUTPUT PARAMETERS:
: 764 1758 1 |     NONE
: 765 1759 1 |
: 766 1760 1 | IMPLICIT OUTPUTS:
: 767 1761 1 |     NONE
: 768 1762 1 |
: 769 1763 1 | ROUTINE VALUE:
: 770 1764 1 |     1 if successful
: 771 1765 1 |     0 if blocks overlap
: 772 1766 1 |
: 773 1767 1 | SIDE EFFECTS:
: 774 1768 1 |     extent cache modified
: 775 1769 1 |
: 776 1770 1 | --
: 777 1771 1 |
: 778 1772 2 BEGIN
: 779 1773 2
: 780 1774 2 LOCAL
: 781 1775 2     EXTENT_CACHE      : REF BBLOCK,      ! pointer to extent cache
: 782 1776 2     EXTENT_LIST      : REF BBLOCKVECTOR [8], ! pointer to extent list
: 783 1777 2     J;                ! extent list index
: 784 1778 2
: 785 1779 2 BIND_COMMON;
: 786 1780 2
: 787 1781 2 ! Search the extent cache until we find an entry whose start LBN is
: 788 1782 2 ! higher than the end LBN of the extent being returned.
: 789 1783 2 !
: 790 1784 2
: 791 1785 2 IF .BLOCK_COUNT LEQ 0
: 792 1786 2 THEN BUG_CHECK (MAPCNTZER, FATAL, 'Attempted to return zero extent to cache');
: 793 1787 2
: 794 1788 2 EXTENT_CACHE = .BBLOCK [.CURRENT_VCB[VCBSL_CACHE], VCASL_EXTCACHE];
: 795 1789 2 EXTENT_LIST = EXTENT_CACHE[VCASQ_EXTLIST];
: 796 1790 2 J = 1;
: 797 1791 2 UNTIL .J GTRU .EXTENT_CACHE[VCASW_EXTCOUNT]
```

```
798 1792 DO
799 1793 BEGIN
800 1794 IF .EXTENT_LIST[J-1, VCASL_EXTLBN] GEQU .START_LBN + .BLOCK_COUNT
801 1795 THEN EXITLOOP;
802 1796 J = .J + 1;
803 1797 END;
804 1798
805 1799 ! If there is a preceding entry, check it for overlap.
806 1800 !
807 1801
808 1802 IF .J GTRU 1
809 1803 THEN
810 1804 BEGIN
811 1805 IF .EXTENT_LIST[J-2, VCASL_EXTLBN] + .EXTENT_LIST[J-2, VCASL_EXTBLOCKS]
812 1806 GTRU .START_LBN
813 1807 THEN RETURN 0;
814 1808 END;
815 1809
816 1810 ! Check for adjacency with the preceding and current extents; if so, do
817 1811 ! a merge.
818 1812 !
819 1813
820 1814 IF .J GTRU 1
821 1815 AND .EXTENT_LIST[J-2, VCASL_EXTLBN] + .EXTENT_LIST[J-2, VCASL_EXTBLOCKS]
822 1816 EQL .START_LBN
823 1817 THEN
824 1818 BEGIN
825 1819 EXTENT_LIST[J-2, VCASL_EXTBLOCKS] = .EXTENT_LIST[J-2, VCASL_EXTBLOCKS] + .BLOCK_COUNT;
826 1820
827 1821 IF .J LEQU .EXTENT_CACHE[VCASW_EXTCOUNT]
828 1822 AND .EXTENT_LIST[J-1, VCASL_EXTLBN] EQL .START_LBN + .BLOCK_COUNT
829 1823 THEN
830 1824 BEGIN
831 1825 EXTENT_LIST[J-2, VCASL_EXTBLOCKS] =
832 1826 .EXTENT_LIST[J-2, VCASL_EXTBLOCKS]
833 1827 + .EXTENT_LIST[J-1, VCASL_EXTBLOCKS];
834 1828 CHSMOVE ((.EXTENT_CACHE[VCASW_EXTCOUNT]-.J)*8,
835 1829 EXTENT_LIST[J, VCASL_EXTBLOCKS],
836 1830 EXTENT_LIST[J-1, VCASL_EXTBLOCKS]);
837 1831 EXTENT_CACHE[VCASW_EXTCOUNT] = .EXTENT_CACHE[VCASW_EXTCOUNT] - 1;
838 1832 END;
839 1833 END
840 1834
841 1835 ELSE IF .J LEQU .EXTENT_CACHE[VCASW_EXTCOUNT]
842 1836 AND .EXTENT_LIST[J-1, VCASL_EXTLBN] EQL .START_LBN + .BLOCK_COUNT
843 1837 THEN
844 1838 BEGIN
845 1839 EXTENT_LIST[J-1, VCASL_EXTBLOCKS] = .EXTENT_LIST[J-1, VCASL_EXTBLOCKS] + .BLOCK_COUNT;
846 1840 EXTENT_LIST[J-1, VCASL_EXTLBN] = .START_LBN;
847 1841 END
848 1842
849 1843 ELSE
850 1844 BEGIN
851 1845 CHSMOVE ((.EXTENT_CACHE[VCASW_EXTCOUNT]-.J+1)*8,
852 1846 EXTENT_LIST[J-1, VCASL_EXTBLOCKS],
853 1847 EXTENT_LIST[J, VCASL_EXTBLOCKS]);
854 1848 EXTENT_LIST[J-1, VCASL_EXTBLOCKS] = .BLOCK_COUNT;
```





			F0 A648 7F 0008D	PUSHAQ	-16(EXTENT_LIST)[J]	1827
			F8 A648 7F 00091	PUSHAQ	-8(EXTENT_LIST)[J]	1828
	9E		9E C0 00095	ADDL2	@(SP)+, @(SP)+	1830
	50		02 A7 3C 00098	MOVZWL	2(EXTENT_CACHE), R0	1831
	50		58 C2 0009C	SUBL2	J, R0	1814
	50		08 C4 0009F	MULL2	#8, R0	1839
			F8 A648 7F 000A2	PUSHAQ	-8(EXTENT_LIST)[J]	1835
			6648 7F 000A6	PUSHAQ	(EXTENT_LIST)[J]	1836
	9E	9E	50 28 000A9	MOV3	R0, @(SP)+, @(SP)+	1839
			02 A7 B7 000AD	DECW	2(EXTENT_CACHE)	1840
			4B 11 000B0	BRB	7\$	1835
			F8 A648 7E 000B2 5\$:	MOVAQ	-8(EXTENT_LIST)[J], R1	1839
58			00 ED 000B7	CMPZV	#0, #16, 2(EXTENT_CACHE), J	1835
			17 1F 000BD	BLSSU	6\$	1836
			FC A648 7F 000BF	PUSHAQ	-4(EXTENT_LIST)[J]	1839
			9E D1 000C3	CMP	@(SP)+, R3	1840
			0E 12 000C6	BNEQ	6\$	1835
			08 AC C0 000C8	ADDL2	BLOCK COUNT, (R1)	1845
			FC A648 7F 000CC	PUSHAQ	-4(EXTENT_LIST)[J]	1847
	9E		04 AC D0 000D0	MOVL	START_LBN, @(SP)+	1848
			27 11 000D4	BRB	7\$	1849
			02 A7 3C 000D6 6\$:	MOVZWL	2(EXTENT_CACHE), R0	1850
			58 C2 000DA	SUBL2	J, R0	1853
			08 C4 000DD	MULL2	#8, R0	1855
			08 C0 000E0	ADDL2	#8, R0	1847
			6648 7F 000E3	PUSHAQ	(EXTENT_LIST)[J]	1848
	9E	61	50 28 000E6	MOV3	R0, (R1), @(SP)+	1849
			F8 A648 7F 000EA	PUSHAQ	-8(EXTENT_LIST)[J]	1850
			08 AC D0 000EE	MOVL	BLOCK COUNT, @(SP)+	1853
			FC A648 7F 000F2	PUSHAQ	-4(EXTENT_LIST)[J]	1855
			04 AC D0 000F6	MOVL	START_LBN, @(SP)+	1857
			02 A7 B6 000FA	INCW	2(EXTENT_CACHE)	1850
04			08 AC C0 000FD 7\$:	ADDL2	BLOCK COUNT, 4(EXTENT_CACHE)	1853
			01 D0 00102	MOVL	#1, R0	1855
			04 04 00105	RET		1857
			50 D4 00106 8\$:	CLRL	R0	1857
			04 04 00108	RET		1857

; Routine Size: 265 bytes, Routine Base: \$CODE\$ + 0488



```

: 865 1858 1 GLOBAL ROUTINE PURGE_EXTENT (ENTRY_COUNT, CACHE_LIMIT) : L_NORM NOVALUE =
: 866 1859 1
: 867 1860 1 |++
: 868 1861 1
: 869 1862 1 FUNCTIONAL DESCRIPTION:
: 870 1863 1
: 871 1864 1 This routine removes the specified number of entries from the
: 872 1865 1 extent cache and returns the blocks to the storage bitmap.
: 873 1866 1
: 874 1867 1
: 875 1868 1 CALLING SEQUENCE:
: 876 1869 1 PURGE_EXTENT (ARG1, ARG2)
: 877 1870 1
: 878 1871 1 INPUT PARAMETERS:
: 879 1872 1 ARG1: number of entries to retain
: 880 1873 1 ARG2: total number of blocks to retain in cache
: 881 1874 1
: 882 1875 1 IMPLICIT INPUTS:
: 883 1876 1 CURRENT_VCB: VCB of volume
: 884 1877 1
: 885 1878 1 OUTPUT PARAMETERS:
: 886 1879 1 NONE
: 887 1880 1
: 888 1881 1 IMPLICIT OUTPUTS:
: 889 1882 1 NONE
: 890 1883 1
: 891 1884 1 ROUTINE VALUE:
: 892 1885 1 NONE
: 893 1886 1
: 894 1887 1 SIDE EFFECTS:
: 895 1888 1 extent cache and storage bitmap modified
: 896 1889 1
: 897 1890 1 --
: 898 1891 1
: 899 1892 2 BEGIN
: 900 1893 2
: 901 1894 2 BUILTIN FP;
: 902 1895 2
: 903 1896 2 LOCAL
: 904 1897 2 EXTENT_CACHE : REF BBLOCK, ! pointer to extent cache
: 905 1898 2 EXTENT_LIST : REF BBLOCKVECTOR [.8], ! pointer to extent list
: 906 1899 2 BLOCK, : bitmap block number of current extent
: 907 1900 2 VBN, : bitmap block number of best group
: 908 1901 2 COUNT, : count of entries in current group
: 909 1902 2 BLOCKS, : block count in current group
: 910 1903 2 BASE_J, : cache index of start of current map block
: 911 1904 2 BEST_COUNT, : count of entries in best group
: 912 1905 2 BEST_BLOCKS, : count of blocks in best group
: 913 1906 2 BEST_J, : index of start of best group
: 914 1907 2 MOST_BLOCKS, : count of blocks in largest group
: 915 1908 2 MOST_J, : starting index on largest group
: 916 1909 2 BLOCKS_TO_REM, : number of blocks to remove from cache
: 917 1910 2 LBN, : starting LBN of extent
: 918 1911 2 BLOCK_COUNT, : count of extent
: 919 1912 2 LOCK_STATUS : VECTOR [2]; : lock status block
: 920 1913 2
: 921 1914 2 BIND_COMMON;

```

```

: 922      1915 2
: 923      1916 2
: 924      1917 2 EXTERNAL ROUTINE
: 925      1918 2     ALLOCATION_LOCK : L_NORM,
: 926      1919 2     ZERO_ON_ERROR;           ! return zero on error signal (handler)
: 927      1920 2 ! Serialize processing against other storage/header allocation/deallocation.
: 928      1921 2 !
: 929      1922 2 !
: 930      1923 2 ALLOCATION_LOCK ();
: 931      1924 2 !
: 932      1925 2 ! If we are not removing all the entries, scan the extent cache for the
: 933      1926 2 ! desired number of entries that reside in the same bitmap block.
: 934      1927 2 !
: 935      1928 2 !
: 936      1929 2 EXTENT_CACHE = .BBLOCK [.CURRENT_VCB[VCBSL_CACHE], VCASL_EXTCACHE];
: 937      1930 2 EXTENT_LIST = EXTENT_CACHE[VCASQ_EXTLIST];
: 938      1931 2 !
: 939      1932 2 IF .ENTRY_COUNT NEQ 0
: 940      1933 2 THEN
: 941      1934 2     BEGIN
: 942      1935 2     BEST_COUNT = 0;
: 943      1936 2     BEST_BLOCKS = 0;
: 944      1937 2     MOST_BLOCKS = 0;
: 945      1938 2     VBN = -1;
: 946      1939 2
: 947      1940 2     INCR J FROM 1 TO .EXTENT_CACHE[VCASW_EXTCOUNT]
: 948      1941 2     DO
: 949      1942 2     BEGIN
: 950      1943 2     BLOCK = (.EXTENT_LIST[J-1, VCASL_EXTLBN] / 4096)
: 951      1944 2     / .CURRENT_VCB[VCBSW_CLUSTER];
: 952      1945 2     IF .BLOCK NEQ .VBN
: 953      1946 2     THEN
: 954      1947 2     BEGIN
: 955      1948 2     VBN = .BLOCK;
: 956      1949 2     COUNT = 0;
: 957      1950 2     BLOCKS = 0;
: 958      1951 2     BASE_J = .J;
: 959      1952 2     END;
: 960      1953 2     COUNT = .COUNT + 1;
: 961      1954 2     BLOCKS = .BLOCKS + .EXTENT_LIST[J-1, VCASL_EXTBLOCKS];
: 962      1955 2
: 963      1956 2     IF .COUNT GTRU .BEST_COUNT
: 964      1957 2     THEN
: 965      1958 2     BEGIN
: 966      1959 2     BEST_COUNT = .COUNT;
: 967      1960 2     BEST_BLOCKS = .BLOCKS;
: 968      1961 2     BEST_J = .BASE_J;
: 969      1962 2     END;
: 970      1963 2
: 971      1964 2     IF .BLOCKS GTRU .MOST_BLOCKS
: 972      1965 2     THEN
: 973      1966 2     BEGIN
: 974      1967 2     MOST_BLOCKS = .BLOCKS;
: 975      1968 2     MOST_J = .BASE_J;
: 976      1969 2     END;
: 977      1970 2     END;
: 978      1971 2

```



```

: 979 1972 : See what we got from scanning the cache. If removing the greatest number
: 980 1973 : of entries will satisfy the space reduction, then do that. Otherwise,
: 981 1974 : go for the set of entries with the most space. If that isn't sufficient,
: 982 1975 : start at the beginning of the cache.
: 983 1976 :
: 984 1977 :
: 985 1978 : BLOCKS_TO_REM = .EXTENT_CACHE[VCSL_EXTTOTAL] - .CACHE_LIMIT;
: 986 1979 : IF .CACHE_LIMIT GTRU .EXTENT_CACHE[VCSL_EXTTOTAL]
: 987 1980 : THEN BLOCKS_TO_REM = 0;
: 988 1981 :
: 989 1982 : IF .BEST_BLOCKS LSSU .BLOCKS_TO_REM
: 990 1983 : THEN
: 991 1984 : BEGIN
: 992 1985 : BEST_J = .MOST_J;
: 993 1986 : IF .MOST_BLOCKS LSSU .BLOCKS_TO_REM
: 994 1987 : THEN BEST_J = 1;
: 995 1988 : END;
: 996 1989 :
: 997 1990 : VBN = (.EXTENT_LIST[BEST_J-1, VCSL_EXTLBN] / 4096) / .CURRENT_VCB[VCSW_CLUSTER];
: 998 1991 :
: 999 1992 : Now scan the extent cache, remove the called for entries, and return
1000 1993 : the blocks to the storage bitmap.
: 1001 1994 :
: 1002 1995 :
: 1003 1996 : UNTIL .BEST_J GTRU .EXTENT_CACHE[VCSW_EXTCOUNT]
: 1004 1997 : DO
: 1005 1998 : BEGIN
: 1006 1999 : LBN = .EXTENT_LIST[BEST_J-1, VCSL_EXTLBN];
: 1007 2000 : IF .EXTENT_CACHE[VCSL_EXTTOTAL] LEQU .CACHE_LIMIT
: 1008 2001 : AND (.EXTENT_CACHE[VCSW_EXTCOUNT] LEQU .ENTRY_COUNT
: 1009 2002 : OR (.VBN NEQ (.LBN / 4096) / .CURRENT_VCB[VCSW_CLUSTER]
: 1010 2003 : AND .ENTRY_COUNT NEQ 0)
: 1011 2004 : )
: 1012 2005 : THEN EXITLOOP;
: 1013 2006 :
: 1014 2007 : BLOCK_COUNT = .EXTENT_LIST[BEST_J-1, VCSL_EXTBLOCKS];
: 1015 2008 : IF .EXTENT_CACHE[VCSL_EXTTOTAL] - .BLOCK_COUNT LSSU .CACHE_LIMIT
: 1016 2009 : AND .EXTENT_CACHE[VCSW_EXTCOUNT] LEQU .ENTRY_COUNT
: 1017 2010 : THEN
: 1018 2011 : BEGIN
: 1019 2012 : BLOCK_COUNT = .EXTENT_CACHE[VCSL_EXTTOTAL] - .CACHE_LIMIT;
: 1020 2013 : BLOCK_COUNT = ((.BLOCK_COUNT + .CURRENT_VCB[VCSW_CLUSTER]-1)
: 1021 2014 : / .CURRENT_VCB[VCSW_CLUSTER]) * .CURRENT_VCB[VCSW_CLUSTER];
: 1022 2015 : END;
: 1023 2016 : REMOVE_EXTENT (.LBN, .BLOCK_COUNT);
: 1024 2017 : RETURN_BITMAP (.LBN, .BLOCK_COUNT);
: 1025 2018 : END;
: 1026 2019 : END
: 1027 2020 :
: 1028 2021 : For a full purge of the extent cache, just sweep through it, releasing
: 1029 2022 : the entries. This is done under a handler so that I/O errors do not
: 1030 2023 : terminate the operation. At the end, we release the cache lock.
: 1031 2024 :
: 1032 2025 :
: 1033 2026 : ELSE
: 1034 2027 : BEGIN
: 1035 2028 : .FP = ZERO_ON_ERROR;

```



```

: 1036      2029      3      UNTIL .EXTENT_CACHE[VCA$W_EXTCOUNT] EQL 0
: 1037      2030      3      DO
: 1038      2031      4      BEGIN
: 1039      2032      4      LBN = .EXTENT_LIST[0, VCA$L_EXTLBN];
: 1040      2033      4      BLOCK_COUNT = .EXTENT_LIST[0, VCA$L_EXTBLOCKS];
: 1041      2034      4      REMOVE_EXTENT (.LBN, .BLOCK_COUNT);
: 1042      2035      4      RETURN_BITMAP (.LBN, .BLOCK_COUNT);
: 1043      2036      4      END;
: 1044      2037      4
: 1045      2038      4      IF .EXTENT_CACHE[VCA$L_EXTCLKID] NEQ 0
: 1046      2039      3      THEN
: 1047      2040      4      BEGIN
: 1048      2041      4      LOCK_STATUS[1] = .EXTENT_CACHE[VCA$L_EXTCLKID];
: 1049      2042      4      IF NOT SENQW (EFN = EFN,
: 1050      2043      4      LKMODE = LCK$K_NLMODE,
: 1051      2044      4      FLAGS = LCK$M_NOQUEUE OR LCK$M_SYNCSTS OR LCK$M_CONVERT OR LCK$M_CVTSYS,
: 1052      2045      4      LKSB = LOCK_STATUS
: 1053      2046      5      )
: 1054      2047      4      THEN BUG_CHECK (XQPERR, FATAL, 'Unexpected lock manager error');
: 1055      2048      3      END;
: 1056      2049      3      BBLOCK [.CURRENT_VCB[VCB$L_CACHE], VCA$V_EXTC_VALID] = 0;
: 1057      2050      2      END;
: 1058      2051      2
: 1059      2052      1      END;

```

! end of routine PURGE\_EXTENT

					.EXTRN	ZERO_ON_ERROR, SYS\$ENQW	
					.EXTRN	BUG\$_XQPERR	
					.ENTRY	PURGE_EXTENT, Save R2,R3,R4,R5,R6,R7,R8,R9,-;	1858
						R11	
					SUBL2	#40, SP	
					MOVAB	-104(BASE), 28(SP)	1912
					CALLS	#0, ALLOCATION_LOCK	1923
					MOVL	@28(SP), R0	1929
					MOVL	88(R0), R0	
					MOVL	4(R0), EXTENT_CACHE	
					MOVAB	44(R2), EXTENT_LIST	1930
					TSTL	ENTRY_COUNT	1932
					BNEQ	1\$	
					BRW	13\$	
					CLRL	BEST_COUNT	1935
					CLRL	BEST_BLOCKS	1936
					CLRL	MOST_BLOCKS	1937
					MNEGL	#1, VBN	1938
					MOVZWL	2(EXTENT_CACHE), 16(SP)	1940
					CLRL	J	1943
					BRB	5\$	
					MOVAQ	(EXTENT_LIST)[J], R0	
					DIVL3	#4096, =4(R0), R8	
					MOVL	@28(SP), R4	1944
					MOVZWL	60(R4), (SP)	
					DIVL3	(SP), R8, BLOCK	
					CMPL	BLOCK, VBN	1945
					BEQL	3\$	
					MOVL	BLOCK, VBN	1948



				56	7C	00061	CLRQ	BLOCKS	1950				
	0C	AE		51	D0	00063	MOVL	J, BASE_J	1951				
				57	D6	00067	INCL	COUNT	1953				
		56	F8	A0	C0	00069	ADDL2	-8(R0), BLOCKS	1954				
		5B		57	D1	0006D	CMPL	COUNT, BEST_COUNT	1956				
				0B	1B	00070	BLEQU	4\$					
		5B		57	D0	00072	MOVL	COUNT, BEST_COUNT	1959				
	18	AE		56	D0	00075	MOVL	BLOCKS, BEST_BLOCKS	1960				
		55	0C	AE	D0	00079	MOVL	BASE_J, BEST_J	1961				
		59		56	D1	0007D	CMPL	BLOCKS, MOST_BLOCKS	1964				
				08	1B	00080	BLEQU	5\$					
		59		56	D0	00082	MOVL	BLOCKS, MOST_BLOCKS	1967				
	04	AE	0C	AE	D0	00085	MOVL	BASE_J, MOST_J	1968				
		51	10	AE	F3	0008A	AOBLEQ	16(SP), J, 2\$	1940				
	04	A2	08	AC	C3	0008F	SUBL3	CACHE_LIMIT, 4(EXTENT_CACHE), BLOCKS_TO_REM	1978				
	04	A2	08	AC	D1	00095	CMPL	CACHE_LIMIT, 4(EXTENT_CACHE)	1979				
				02	1B	0009A	BLEQU	6\$					
				50	D4	0009C	CLRL	BLOCKS_TO_REM	1980				
		50	18	AE	D1	0009E	CMPL	BEST_BLOCKS, BLOCKS_TO_REM	1982				
				0C	1E	000A2	BGEQU	7\$					
		55	04	AE	D0	000A4	MOVL	MOST_J, BEST_J	1985				
		50		59	D1	000A8	CMPL	MOST_BLOCKS, -BLOCKS_TO_REM	1986				
				03	1E	000AB	BGEQU	7\$					
		55		01	D0	000AD	MOVL	#1, BEST_J	1987				
			FC	A345	7F	000B0	PUSHAQ	-4(EXTENT_LIST)[BEST_J]	1990				
		50	00001000	8F	C7	000B4	DIVL3	#4096, @28(SP)+, R0					
				1C	BE	000BC	MOVL	@28(SP), R1					
				3C	A1	000C0	MOVZWL	60(R1), R4					
				54	C7	000C4	DIVL3	R4, R0, VBN					
	55	14	AE	50		00	ED	000C9	8\$:				
		02	A2	10		01	1E	000CF	BGEQU	9\$	1996		
						04	000D1	RET					
				50	6345	7E	000D2	MOVAQ	(EXTENT_LIST)[BEST_J], R0	1999			
				56	FC	A0	D0	000D6	MOVL	-4(R0), LBN			
				08	04	A2	D1	000DA	CMPL	4(EXTENT_CACHE), CACHE_LIMIT	2000		
						29	1A	000DF	BGTRU	11\$			
	04	AC	02	A2	10	00	ED	000E1	CMPZV	#0, #16, 2(EXTENT_CACHE), ENTRY_COUNT	2001		
						01	1A	000E8	BGTRU	10\$			
						04	000EA	RET					
				54	00001000	8F	C7	000EB	DIVL3	#4096, LBN, R4	2002		
					1C	BE	D0	000F3	MOVL	@28(SP), R1			
					3C	A1	3C	000F7	MOVZWL	60(R1), R7			
						57	C6	000FB	DIVL2	R7, R4			
				54	14	AE	D1	000FE	CMPL	VBN, R4			
						06	13	00102	BEQL	11\$			
					04	AC	D5	00104	TSTL	ENTRY_COUNT	2003		
						01	13	00107	BEQL	11\$			
						04	00109	RET					
				58	F8	A0	D0	0010A	MOVL	-8(R0), BLOCK_COUNT	2007		
						58	C3	0010E	SUBL3	BLOCK_COUNT, 4(EXTENT_CACHE), R0	2008		
		50	04	A2		50	D1	00113	CMPL	R0, CACHE_LIMIT			
			08	AC		25	1E	00117	BGEQU	12\$			
						00	ED	00119	CMPZV	#0, #16, 2(EXTENT_CACHE), ENTRY_COUNT	2009		
	04	AC	02	A2	10	1C	1A	00120	BGTRU	12\$			
						01	1A	00120	BGTRU	12\$			
				58	04	A2	08	AC	C3	00122	SUBL3	CACHE_LIMIT, 4(EXTENT_CACHE), BLOCK_COUNT	2012
						50	1C	BE	D0	00128	MOVL	@28(SP), R0	2013
						50	3C	C0	0012C	ADDL2	#60, R0		



	50		60	3C	0012F		MOVZWL	(R0), R0			
	51	FF	A048	9E	00132		MOVAB	-1(R0)[BLOCK_COUNT], R1			
58	51		50	C6	00137		DIVL2	R0, R1	2014		
	51		50	C5	0013A		MULL3	R0, R1, BLOCK_COUNT			
	0000V	CF	0140	8F	BB	0013E	12\$:	PUSHR	#^M<R6,R8>	2016	
				02	FB	00142		CALLS	#2, REMOVE_EXTENT		
	0000V	CF	0140	8F	BB	00147		PUSHR	#^M<R6,R8>	2017	
				02	FB	0014B		CALLS	#2, RETURN_BITMAP		
				FF	76	31	00150	BRW	8\$	1996	
	6D	0000G	02	CF	9E	00153	13\$:	MOVAB	ZERO_ON_ERROR, (FP)	2028	
				A2	B5	00158	14\$:	TSTW	2(EXTENT_CACHE)	2029	
				1B	13	0015B		BEQL	15\$		
	56		04	A3	D0	0015D		MOVL	4(EXTENT_LIST), LBN	2032	
	58			63	D0	00161		MOVL	(EXTENT_LIST), BLOCK_COUNT	2033	
	0000V	CF	0140	8F	BB	00164		PUSHR	#^M<R6,R8>	2034	
				02	FB	00168		CALLS	#2, REMOVE_EXTENT		
	0000V	CF	0140	8F	BB	0016D		PUSHR	#^M<R6,R8>	2035	
				02	FB	00171		CALLS	#2, RETURN_BITMAP		
				E0	11	00176		BRB	14\$	2029	
				0C	A2	D5	00178	15\$:	TSTL	12(EXTENT_CACHE)	2038
				25	13	0017B		BEQL	16\$		
	24	AE	0C	A2	D0	0017D		MOVL	12(EXTENT_CACHE), LOCK_STATUS+4	2041	
				7E	7C	00182		CLRQ	-(SP)	2046	
				7E	7C	00184		CLRQ	-(SP)		
				7E	7C	00186		CLRQ	-(SP)		
				7E	D4	00188		CLRL	-(SP)		
	7E	4E	40	8F	9A	0018A		MOVZBL	#78, -(SP)		
				AE	9F	0018E		PUSHAB	LOCK_STATUS		
	7E			1E	7D	00191		MOVQ	#30, -(SP)		
	00000000G	00		0B	FB	00194		CALLS	#11, SYS\$ENQW		
		04		50	E8	0019B		BLBS	R0, 16\$		
				FE	FF	0019E		BUGW		2047	
				0000*	001A0			.WORD	<BUG\$ XQPERR!4>		
	50	1C	58	BE	D0	001A2	16\$:	MOVL	@28(SP), R0	2049	
	50			A0	D0	001A6		MOVL	88(R0), R0		
	0B	A0		02	8A	001AA		BICB2	#2, 11(R0)		
				04	001AE			RET		2052	

; Routine Size: 431 bytes, Routine Base: \$CODE\$ + 0591



```
: 1061      2053 1 ROUTINE REMOVE_EXTENT (LBN, COUNT) : L_NORM =
: 1062      2054 1
: 1063      2055 1 !++
: 1064      2056 1
: 1065      2057 1 FUNCTIONAL DESCRIPTION:
: 1066      2058 1
: 1067      2059 1     This routine removes the indicated number of blocks from the indicated
: 1068      2060 1     extent in the cache. If the total block count of the extent is removed,
: 1069      2061 1     then the extent is eliminated completely.
: 1070      2062 1
: 1071      2063 1
: 1072      2064 1 CALLING SEQUENCE:
: 1073      2065 1     REMOVE_EXTENT (ARG1, ARG2)
: 1074      2066 1
: 1075      2067 1 INPUT PARAMETERS:
: 1076      2068 1     ARG1: LBN of extent to remove
: 1077      2069 1     ARG2: count of blocks to remove
: 1078      2070 1
: 1079      2071 1 IMPLICIT INPUTS:
: 1080      2072 1     CURRENT_VCB: VCB of volume
: 1081      2073 1
: 1082      2074 1 OUTPUT PARAMETERS:
: 1083      2075 1     NONE
: 1084      2076 1
: 1085      2077 1 IMPLICIT OUTPUTS:
: 1086      2078 1     NONE
: 1087      2079 1
: 1088      2080 1 ROUTINE VALUE:
: 1089      2081 1     1
: 1090      2082 1
: 1091      2083 1 SIDE EFFECTS:
: 1092      2084 1     extent cache altered
: 1093      2085 1
: 1094      2086 1 !--
: 1095      2087 1
: 1096      2088 2 BEGIN
: 1097      2089 2
: 1098      2090 2 LOCAL
: 1099      2091 2     EXTENT_CACHE      : REF BBLOCK,      ! pointer to extent cache
: 1100      2092 2     EXTENT_LIST       : REF BBLOCKVECTOR [.8]; ! pointer to extent list
: 1101      2093 2
: 1102      2094 2 BIND_COMMON;
: 1103      2095 2
: 1104      2096 2 ! Get the pointer to the extent cache and search it for the LBN. When
: 1105      2097 2 ! found, squish out the entry.
: 1106      2098 2 !
: 1107      2099 2 !
: 1108      2100 2 EXTENT_CACHE = .BBLOCK [.CURRENT_VCB[VCBSL_CACHE], VCASL_EXTCACHE];
: 1109      2101 2 EXTENT_LIST = EXTENT_CACHE[VCASQ_EXTLIST];
: 1110      2102 2
: 1111      2103 2 INCR J FROM 1 TO .EXTENT_CACHE[VCASW_EXTCOUNT]
: 1112      2104 2 DO
: 1113      2105 2     BEGIN
: 1114      2106 2     IF .EXTENT_LIST[J-1, VCASL_EXTLBN] EQL .LBN
: 1115      2107 2     THEN
: 1116      2108 2         BEGIN
: 1117      2109 4         EXTENT_LIST[J-1, VCASL_EXTLBN] = .EXTENT_LIST[J-1, VCASL_EXTLBN] + .COUNT;
```

```

: 1118      2110  4      EXTENT_LIST[J-1, VCASL_EXTBLOCKS] = .EXTENT_LIST[J-1, VCASL_EXTBLOCKS] - .COUNT;
: 1119      2111  4      IF .EXTENT_LIST[J-1, VCASL_EXTBLOCKS] NEQ 0 THEN EXITLOOP;
: 1120      2112  4      CH$MOVE ((.EXTENT_CACHE[VCASW_EXTCOUNT]-J)*8,
: 1121      2113  4          EXTENT_LIST[J, VCASL_EXTBLOCKS],
: 1122      2114  4          EXTENT_LIST[J-1, VCASL_EXTBLOCKS]);
: 1123      2115  4      EXTENT_CACHE[VCASW_EXTCOUNT] = .EXTENT_CACHE[VCASW_EXTCOUNT] - 1;
: 1124      2116  4      EXITLOOP;
: 1125      2117  4      END;
: 1126      2118  3      END;
: 1127      2119  2      END;
: 1128      2120  2      EXTENT_CACHE[VCASL_EXTTOTAL] = .EXTENT_CACHE[VCASL_EXTTOTAL] - .COUNT;
: 1129      2121  2      1
: 1130      2122  1
: 1131      2123  1      END;

```

! end of routine REMOVE\_EXTENT

		03FC 0000 REMOVE_EXTENT:				
				.WORD	Save R2,R3,R4,R5,R6,R7,R8,R9	2053
50	98	AA	D0 00002	MOVL	-104(BASE), R0	2100
50	58	A0	D0 00006	MOVL	88(R0), R0	
57	04	A0	D0 0000A	MOVL	4(R0), EXTENT_CACHE	
56	2C	A7	9E 0000E	MOVAB	44(R7), EXTENT_LIST	2101
59	02	A7	3C 00012	MOVZWL	2(EXTENT_CACHE), R9	2103
			58 D4 00016	CLRL	J	
			3C 11 00018	BRB	2\$	
		FC	A648 7F 0001A 1\$:	PUSHAQ	-4(EXTENT_LIST)[J]	2106
04	AC		9E D1 0001E	CPL	@(SP)+, LBN	
			32 12 00022	BNEQ	2\$	
		FC	A648 7F 00024	PUSHAQ	-4(EXTENT_LIST)[J]	2109
9E		08	AC C0 00028	ADDL2	COUNT, @(SP)+	
		F8	A648 7F 0002C	PUSHAQ	-8(EXTENT_LIST)[J]	2110
9E		08	AC C2 00030	SUBL2	COUNT, @(SP)+	
		F8	A648 7F 00034	PUSHAQ	-8(EXTENT_LIST)[J]	2111
			9E D5 00038	TSTL	@(SP)+	
			1E 12 0003A	BNEQ	3\$	
50	02	A7	3C 0003C	MOVZWL	2(EXTENT_CACHE), R0	2112
50			58 C2 00040	SUBL2	J, R0	
50			08 C4 00043	MULL2	#8, R0	
		F8	A648 7F 00046	PUSHAQ	-8(EXTENT_LIST)[J]	2114
			6648 7F 0004A	PUSHAQ	(EXTENT_LIST)[J]	
9E	9E		50 28 0004D	MOV3	R0, @(SP)+, @(SP)+	
		02	A7 B7 00051	DECW	2(EXTENT_CACHE)	2115
			04 11 00054	BRB	3\$	2108
CO			58 59 F3 00056 2\$:	AOBLEQ	R9, J, 1\$	2103
	04	08	AC C2 0005A 3\$:	SUBL2	COUNT, 4(EXTENT_CACHE)	2120
			50 01 D0 0005F	MOVL	#1, R0	2123
			04 00062	RET		

; Routine Size: 99 bytes, Routine Base: \$CODE\$ + 0740



```

: 1133 2124 1 ROUTINE ALLOC_BITMAP (FIB, BLOCKS_NEEDED, START_LBN, BLOCKS_ALLOC, PARTIAL) : L_NORM =
: 1134 2125 1
: 1135 2126 1 ++
: 1136 2127 1
: 1137 2128 1 FUNCTIONAL DESCRIPTION:
: 1138 2129 1
: 1139 2130 1 This routine allocates a single contiguous area of disk.
: 1140 2131 1 Mode of allocation is determined by the allocation control
: 1141 2132 1 in the FIB.
: 1142 2133 1
: 1143 2134 1 CALLING SEQUENCE:
: 1144 2135 1 ALLOC_BITMAP (ARG1, ARG2, ARG3, ARG4, ARG5)
: 1145 2136 1
: 1146 2137 1 INPUT PARAMETERS:
: 1147 2138 1 ARG1: address of FIB for this operation
: 1148 2139 1 ARG2: number of blocks to allocate
: 1149 2140 1 ARG5: 0 to scan entire bitmap
: 1150 2141 1 1 to scan only currently resident block
: 1151 2142 1
: 1152 2143 1 IMPLICIT INPUTS:
: 1153 2144 1 CURRENT_VCB: ADDRESS OF VCB IN PROCESS
: 1154 2145 1 CURRENT_UCB: ADDRESS OF UCB IN PROCESS
: 1155 2146 1
: 1156 2147 1 OUTPUT PARAMETERS:
: 1157 2148 1 ARG3: address of longword to store starting LBN
: 1158 2149 1 ARG4: address of longword to store block count
: 1159 2150 1
: 1160 2151 1 IMPLICIT OUTPUTS:
: 1161 2152 1 LOC_LBN: placement LBN of allocation or 0
: 1162 2153 1 NONE
: 1163 2154 1
: 1164 2155 1 ROUTINE VALUE:
: 1165 2156 1 1 if successful allocation
: 1166 2157 1 0 if failure
: 1167 2158 1
: 1168 2159 1 SIDE EFFECTS:
: 1169 2160 1 storage map and VCB modified
: 1170 2161 1
: 1171 2162 1 --
: 1172 2163 1
: 1173 2164 2 BEGIN
: 1174 2165 2
: 1175 2166 2 BUILTIN
: 1176 2167 2 EDIV;
: 1177 2168 2
: 1178 2169 2 MAP
: 1179 2170 2 FIB : REF BBLOCK; ! FIB of request
: 1180 2171 2
: 1181 2172 2 LOCAL
: 1182 2173 2 CLUSTER, ! cluster factor of volume
: 1183 2174 2 QUAD_BLOCKS_NEEDED : VECTOR [2], ! Blocks needed as a quadword
: 1184 2175 2 BITS_NEEDED, ! number of map bits to allocate
: 1185 2176 2 BEGIN_BIT, ! first bitmap bit looked at
: 1186 2177 2 START_BIT, ! bit address in storage map
: 1187 2178 2 BIT_COUNT, ! number of bits to scan
: 1188 2179 2 FIRST_SET, ! start of free area
: 1189 2180 2 BITS_SCANNED, ! number of bits processed by scanner

```

```

: 1190      2181 2      END BIT,
: 1191      2182 2      BEST_STARTBIT,
: 1192      2183 2      BEST_BITSFIND,
: 1193      2184 2      CYL_SIZE,
: 1194      2185 2      CYL_BOUNDARY,
: 1195      2186 2      DUMMY;
: 1196      2187 2
: 1197      2188 2 LABEL
: 1198      2189 2      MAP_SCAN;
: 1199      2190 2
: 1200      2191 2 BIND_COMMON;
: 1201      2192 2
: 1202      2193 2 ! Adjust the desired block count to a bit count through the volume
: 1203      2194 2 ! cluster factor. Set up the running parameters.
: 1204      2195 2
: 1205      2196 2
: 1206      2197 2 CLUSTER = .CURRENT_VCB[VCSW_CLUSTER];
: 1207      2198 2 QUAD_BLOCKS_NEEDED[0] = .BLOCKS_NEEDED + .CLUSTER - 1;
: 1208      2199 2 QUAD_BLOCKS_NEEDED[1] = 0;
: 1209      2200 2 EDIV (CLUSTER, QUAD_BLOCKS_NEEDED, BITS_NEEDED, DUMMY);
: 1210      2201 2 BEST_BITSFIND = 0;
: 1211      2202 2 START_BIT = BEGIN_BIT = .CURRENT_VCB[VCSB_SMAPVBN] * 4096;
: 1212      2203 2
: 1213      2204 2 CYL_SIZE = .CURRENT_UCB[UCSB_SECTORS]
: 1214      2205 2 * .CURRENT_UCB[UCSB_TRACKS]
: 1215      2206 2 / .CURRENT_VCB[VCSB_BLOCKFACT];
: 1216      2207 2
: 1217      2208 2 ! Get placement data if specified. If the placement LBN is garbage, fail if
: 1218      2209 2 ! exact placement is called for, else forget it.
: 1219      2210 2
: 1220      2211 2
: 1221      2212 2 IF .LOC_LBN NEQ 0
: 1222      2213 2 THEN
: 1223      2214 2 BEGIN
: 1224      2215 2 IF .LOC_LBN GEQU .CURRENT_UCB[UCB$L_MAXBLOCK]
: 1225      2216 2 THEN
: 1226      2217 2 BEGIN
: 1227      2218 2 IF .FIB[FIB$V_EXACT]
: 1228      2219 2 THEN RETURN 0
: 1229      2220 2 ELSE LOC_LBN = 0;
: 1230      2221 2 END;
: 1231      2222 2 START_BIT = BEGIN_BIT = .LOC_LBN / .CLUSTER;
: 1232      2223 2 END;
: 1233      2224 2
: 1234      2225 2 ! The outer loop potentially scans the map twice: once from the given starting
: 1235      2226 2 ! point through to the end and then from beginning to end, if necessary to
: 1236      2227 2 ! locate a large contiguous area with a bad start.
: 1237      2228 2
: 1238      2229 2
: 1239      2230 2 MAP_SCAN:
: 1240      2231 2 BEGIN
: 1241      2232 2 WHILE 1 DO
: 1242      2233 2 BEGIN
: 1243      2234 2 BIT_COUNT = .CURRENT_UCB[UCB$L_MAXBLOCK] / .CLUSTER - .START_BIT;
: 1244      2235 2 IF .PARTIAL
: 1245      2236 2 THEN BIT_COUNT = MINU (.BIT_COUNT, 4096);
: 1246      2237 2
```



```

: 1247      2238  4  ! Now scan the bitmap for the first free block. Having found it, scan
: 1248      2239  4  ! to see how many free blocks there are there. If it is a non-contiguous
: 1249      2240  4  ! allocation, accept the blocks regardless. If it is contiguous, and the
: 1250      2241  4  ! free area is too small, keep looking.
: 1251      2242  4  !
: 1252      2243  4  !
: 1253      2244  4  !
: 1254      2245  5  !
: 1255      2246  5  !
: 1256      2247  5  !
: 1257      2248  5  !
: 1258      2249  6  !
: 1259      2250  6  !
: 1260      2251  6  !
: 1261      2252  6  !
: 1262      2253  6  !
: 1263      2254  6  !
: 1264      2255  5  !
: 1265      2256  5  !
: 1266      2257  5  !
: 1267      2258  5  !
: 1268      2259  5  !
: 1269      2260  5  !
: 1270      2261  5  !
: 1271      2262  5  !
: 1272      2263  5  !
: 1273      2264  5  !
: 1274      2265  5  !
: 1275      2266  5  !
: 1276      2267  6  !
: 1277      2268  6  !
: 1278      2269  6  !
: 1279      2270  6  !
: 1280      2271  7  !
: 1281      2272  7  !
: 1282      2273  7  !
: 1283      2274  7  !
: 1284      2275  7  !
: 1285      2276  7  !
: 1286      2277  7  !
: 1287      2278  7  !
: 1288      2279  7  !
: 1289      2280  7  !
: 1290      2281  6  !
: 1291      2282  5  !
: 1292      2283  5  !
: 1293      2284  5  !
: 1294      2285  5  !
: 1295      2286  5  !
: 1296      2287  5  !
: 1297      2288  5  !
: 1298      2289  5  !
: 1299      2290  5  !
: 1300      2291  6  !
: 1301      2292  6  !
: 1302      2293  6  !
: 1303      2294  5  !

      WHILE 1 DO
      BEGIN
        IF .LOC_LBN EQL 0
        THEN
          BEGIN
            IF BITSCAN (FIND_SET, .START_BIT, .BIT_COUNT, FIRST_SET, BITS_SCANNED)
            THEN EXITLOOP;          ! out if end of map

            BIT_COUNT = .BIT_COUNT - .BITS_SCANNED;
          END
        ELSE
          FIRST_SET = .START_BIT;

        ! If on cylinder allocation is requested, see if sufficient space remains
        ! between the current point and the next cylinder boundary. If not, nudge
        ! to the next cylinder boundary if exact is not specified. If exact is
        ! specified, we allow for a nudge of 1 cluster to allow for the vagaries
        ! of cluster boundaries.

        IF .FIB[FIBSV_ONCYL]
        THEN
          BEGIN
            CYL_BOUNDARY = ((.FIRST_SET*.CLUSTER) /.CYL_SIZE + 1) * .CYL_SIZE;
            IF .CYL_BOUNDARY/.CLUSTER - .FIRST_SET LEQU .BITS_NEEDED
            THEN
              BEGIN
                CYL_BOUNDARY = (.CYL_BOUNDARY + .CLUSTER - 1) / .CLUSTER;
                IF .FIB[FIBSV_EXACT]
                AND .LOC_LBN NEQ 0
                AND .CYL_BOUNDARY - .FIRST_SET GTRU 1
                THEN RETURN 0;

                BIT_COUNT = .BIT_COUNT - .CYL_BOUNDARY + .FIRST_SET;
                IF .BIT_COUNT LEQ 0 THEN EXITLOOP;
                FIRST_SET = .CYL_BOUNDARY;
              END;
            END;

            BITSCAN (FIND_CLEAR, .FIRST_SET, MIN (.BIT_COUNT, .BITS_NEEDED),
                    START_BIT, BITS_SCANNED);

            BIT_COUNT = .BIT_COUNT - .BITS_SCANNED;

            IF .BITS_SCANNED GTRU .BEST_BITSFIND
            THEN
              BEGIN
                BEST_STARTBIT = .FIRST_SET;
                BEST_BITSFIND = .BITS_SCANNED;
              END;
          END;
      END;
```





				OBFC 00000		ALLOC_BITMAP:			
		5E		1C	C2	00002	.WORD	Save R2,R3,R4,R5,R6,R7,R8,R9,R11	2124
		56		AA	9E	00005	SUBL2	#28, SP	
		59		AA	9E	00009	MOVAB	-128(BASE), R6	2189
		50		AA	D0	0000D	MOVAB	32(BASE), R9	
		54		3C	A0	00011	MOVL	-104(BASE), R0	2197
50		54		08	AC	C1	MOVZWL	60(R0), CLUSTER	
	14	AE		FF	A0	9E	ADDL3	BLOCKS_NEEDED, CLUSTER, R0	2198
				18	AE	D4	MOVAB	-1(R0), QUAD_BLOCKS_NEEDED	
50	58	14	AE	54	7B	00022	CLRL	QUAD_BLOCKS_NEEDED+4	2199
							EDIV	CLUSTER, QUAD_BLOCKS_NEEDED, BITS_NEEDED, -	2200
								DUMMY	
		50		55	D4	00028	CLRL	BEST_BITSFOUND	2201
		57		AA	D0	0002A	MOVL	-104(BASE), R0	2202
57		57		3B	A0	9A	MOVZBL	59(R0), BEGIN_BIT	
	08	AE		0C	78	00032	ASHL	#12, BEGIN_BIT, BEGIN_BIT	
		50		57	D0	00036	MOVL	BEGIN_BIT, START_BIT	
		51		94	AA	D0	MOVL	-108(BASE), R0	2204
		50		44	A0	9A	MOVZBL	68(R0), R1	2205
		50		45	A0	9A	MOVZBL	69(R0), R0	
		51		51	C4	00046	MULL2	R1, R0	
		58		98	AA	D0	MOVL	-104(BASE), R1	2206
58		50		52	A1	9A	MOVZBL	82(R1), CYL_SIZE	
					5B	C7	DIVL3	CYL_SIZE, R0, CYL_SIZE	
					69	D5	TSTL	(R9)	2212
					20	13	BEQL	3\$	
		50		94	AA	D0	MOVL	-108(BASE), R0	2215
	00B0	C0			69	D1	CML	(R9), 176(R0)	
		50			0D	1F	BLSSU	2\$	
		03			04	AC	MOVL	FIB, R0	2218
					20	A0	BLBC	32(R0), 1\$	
					015F	31	BRW	20\$	
					69	D4	CLRL	(R9)	2220
57		69			54	C7	DIVL3	CLUSTER, (R9), BEGIN_BIT	2222
	08	AE			57	D0	MOVL	BEGIN_BIT, START_BIT	
		50			94	AA	MOVL	-108(BASE), R0	2234
50	00B0	C0			54	C7	DIVL3	CLUSTER, 176(R0), R0	
53		50			08	AE	SUBL3	START_BIT, R0, BIT_COUNT	
		14			14	AC	BLBC	PARTIAL, 5\$	2235
		50			53	D0	MOVL	BIT_COUNT, R0	2234
	00001000	8F			50	D1	CML	R0, #4096	
		50			05	1B	BLEQU	4\$	
		53			8F	3C	MOVZWL	#4096, R0	
					50	D0	MOVL	R0, BIT_COUNT	
					69	D5	TSTL	(R9)	2247
					1E	12	BNEQ	7\$	
					0C	AE	PUSHAB	BITS_SCANNED	2250
					08	AE	PUSHAB	FIRST_SET	
					53	DD	PUSHL	BIT_COUNT	
					14	AE	PUSHL	START_BIT	
					7E	D4	CLRL	-(SP)	
	0000V	CF			05	FB	CALLS	#5, BITSCAN	

		03	50	E9	000B6		BLBC	R0, 6\$		
			00B4	31	000B9		BRW	16\$		
		53	0C	AE	C2	000BC	6\$:	SUBL2	BITS_SCANNED, BIT_COUNT	2253
				05	11	000C0		BRB	8\$	2247
	04	AE	08	AE	D0	000C2	7\$:	MOVL	START_BIT, FIRST_SET	2256
45	20	50	04	AC	D0	000C7	8\$:	MOVL	FIB, R0	2265
51	04	AE		01	E1	000CB		BBC	#1, 32(R0), 10\$	
		51		54	C5	000D0		MULL3	CLUSTER, FIRST_SET, R1	2268
				5B	C6	000D5		DIVL2	CYL_SIZE, R1	
				51	D6	000D8		INCL	R1	
52		51		5B	C5	000DA		MULL3	CYL_SIZE, R1, CYL_BOUNDARY	
51		52		54	C7	000DE		DIVL3	CLUSTER, CYL_BOUNDARY, R1	2269
		51	04	AE	C2	000E2		SUBL2	FIRST_SET, RT	
		58		51	D1	000E6		CMPL	R1, BITS_NEEDED	
				2A	1A	000E9		BGTRU	10\$	
		51	FF	A442	9E	000EB		MOVAB	-1(CLUSTER)[CYL_BOUNDARY], R1	2272
52		51		54	C7	000F0		DIVL3	CLUSTER, R1, CYL_BOUNDARY	
		0E	20	A0	E9	000F4		BLBC	32(R0), 9\$	2273
				69	D5	000F8		TSTL	(R9)	2274
				0A	13	000FA		BEQL	9\$	
51		52	04	AE	C3	000FC		SUBL3	FIRST_SET, CYL_BOUNDARY, R1	2275
		01		51	D1	00101		CMPL	R1, #T	
				63	1A	00104		BGTRU	14\$	
50		53		52	C3	00106	9\$:	SUBL3	CYL_BOUNDARY, BIT_COUNT, R0	2278
53		50	04	AE	C1	0010A		ADDL3	FIRST_SET, R0, BIT_COUNT	
				5F	15	0010F		BLEQ	16\$	2279
	04	AE		52	D0	00111		MOVL	CYL_BOUNDARY, FIRST_SET	2280
			0C	AE	9F	00115	10\$:	PUSHAB	BITS_SCANNED	2284
			0C	AE	9F	00118		PUSHAB	START_BIT	
				53	DD	0011B		PUSHL	BIT_COUNT	
		58		6E	D1	0011D		CMPL	(SP), BITS_NEEDED	
				03	15	00120		BLEQ	11\$	
		6E		58	D0	00122		MOVL	BITS_NEEDED, (SP)	
			1C	AE	DD	00125	11\$:	PUSHL	FIRST_SET	
				01	DD	00128		PUSHL	#1	
	0000V	CF		05	FB	0012A		CALLS	#5, BITSCAN	
		53	0C	AE	C2	0012F		SUBL2	BITS_SCANNED, BIT_COUNT	2287
		55	0C	AE	D1	00133		CMPL	BITS_SCANNED, BEST_BITSFOUND	2289
				08	1B	00137		BLEQU	12\$	
		6E	04	AE	D0	00139		MOVL	FIRST_SET, BEST_STARTBIT	2292
		55	0C	AE	D0	0013D		MOVL	BITS_SCANNED, BEST_BITSFOUND	2293
		58		55	D1	00141	12\$:	CMPL	BEST_BITSFOUND, BITS_NEEDED	2296
				3A	1E	00144		BGEQU	17\$	
		50	04	AC	D0	00146		MOVL	FIB, R0	2297
		09	16	A0	E8	0014A		BLBS	22(R0), 13\$	
04	16	A0		01	E0	0014E		BBS	#1, 22(R0), 13\$	
				55	D5	00153		TSTL	BEST_BITSFOUND	2298
				29	12	00155		BNEQ	17\$	
				53	D5	00157	13\$:	TSTL	BIT_COUNT	2301
				15	13	00159		BEQL	16\$	
		50	04	AC	D0	0015B		MOVL	FIB, R0	2308
		08	16	A0	E9	0015F		BLBC	22(R0), 15\$	
		04	20	A0	E9	00163		BLBC	32(R0), 15\$	2309
				69	D5	00167		TSTL	(R9)	2310
				63	12	00169	14\$:	BNEQ	20\$	
				69	D4	0016B	15\$:	CLRL	(R9)	2312
			FF	30	31	0016D		BRW	5\$	2244



				57	D5	00170	16\$:	TSTL	BEGIN_BIT	:	2320		
				0C	13	00172		BEQL	17\$	:			
		08		14	AC	E8	00174	BLBS	PARTIAL, 17\$	:	2321		
				08	AE	D4	00178	CLRL	START_BIT	:	2323		
					57	D4	0017B	CLRL	BEGIN_BIT	:			
					FEF9	31	0017D	BRW	3\$	:	2232		
					55	D5	00180	17\$:	TSTL	BEST_BITSFIND	:	2333	
					12	13	00182	BEQL	18\$	:			
		50		04	AC	D0	00184	MOVL	FIB, R0	:	2334		
		11		16	A0	E9	00188	BLBC	22(R0), 19\$	:			
	0C	16			01	E0	0018C	BBS	#1, 22(R0), 19\$	:			
					55	D1	00191	CMPL	BEST_BITSFIND, BITS_NEEDED	:	2335		
					07	1E	00194	BGEQU	19\$	:			
04	A6			55	54	C5	00196	18\$:	MULL3	CLUSTER, BEST_BITSFIND, 4(R6)	:	2338	
					31	11	0019B	BRB	20\$	:	2339		
					0C	AE	9F	0019D	19\$:	PUSHAB	BITS_SCANNED	:	2342
					14	AE	9F	001A0	PUSHAB	END_BIT	:		
					55	DD	001A3	PUSHL	BEST_BITSFIND	:			
					0C	AE	DD	001A5	PUSHL	BEST_STARTBIT	:		
					03	DD	001A8	PUSHL	#3	:			
		0000V	CF		05	FB	001AA	CALLS	#5, BITSCAN	:			
			50	98	AA	D0	001AF	MOVL	-104(BASE), R0	:	2344		
	51	10	AE	00001000	8F	C7	001B3	DIVL3	#4096, END_BIT, R1	:			
		3B	A0		51	90	001BC	MOVB	R1, 59(R0)	:			
0C	BC		6E		54	C5	001C0	MULL3	CLUSTER, BEST_STARTBIT, @START_LBN	:	2346		
10	BC		55		54	C5	001C5	MULL3	CLUSTER, BEST_BITSFIND, @BLOCKS_ALLOC	:	2347		
			50		01	D0	001CA	MOVL	#1, R0	:	2349		
						04	001CD	RET		:			
					50	D4	001CE	20\$:	CLRL	R0	:	2351	
					04	001D0		RET		:			

: Routine Size: 465 bytes, Routine Base: \$CODE\$ + 07A3

```

: 1362 2352 1 ROUTINE RETURN_BITMAP (START_LBN, BLOCK_COUNT) : L_NORM NOVALUE =
: 1363 2353 1
: 1364 2354 1 |++
: 1365 2355 1 |
: 1366 2356 1 | FUNCTIONAL DESCRIPTION:
: 1367 2357 1 |
: 1368 2358 1 |     This routine returns a single contiguous area to the storage map.
: 1369 2359 1 |
: 1370 2360 1 | CALLING SEQUENCE:
: 1371 2361 1 |     RETURN_BITMAP (ARG1, ARG2)
: 1372 2362 1 |
: 1373 2363 1 | INPUT PARAMETERS:
: 1374 2364 1 |     ARG1: starting LBN to free
: 1375 2365 1 |     ARG2: number of blocks to free
: 1376 2366 1 |
: 1377 2367 1 | IMPLICIT INPUTS:
: 1378 2368 1 |     CURRENT_VCB: VCB of volume
: 1379 2369 1 |     CURRENT_UCB: UCB of device
: 1380 2370 1 |
: 1381 2371 1 | OUTPUT PARAMETERS:
: 1382 2372 1 |     NONE
: 1383 2373 1 |
: 1384 2374 1 | IMPLICIT OUTPUTS:
: 1385 2375 1 |     NONE
: 1386 2376 1 |
: 1387 2377 1 | ROUTINE VALUE:
: 1388 2378 1 |     NONE
: 1389 2379 1 |
: 1390 2380 1 | SIDE EFFECTS:
: 1391 2381 1 |     storage map and VCB modified
: 1392 2382 1 |
: 1393 2383 1 | --
: 1394 2384 1 |
: 1395 2385 2 BEGIN
: 1396 2386 2
: 1397 2387 2 LOCAL
: 1398 2388 2     START_BIT,           ! starting bit number in storage map
: 1399 2389 2     BIT_COUNT,         ! number of bits to set
: 1400 2390 2     DUMMY1,           ! dummies to receive return data
: 1401 2391 2     DUMMY2;         ! from BITSCAN, which is not used
: 1402 2392 2
: 1403 2393 2 BIND_COMMON;
: 1404 2394 2
: 1405 2395 2 ! First check the blocks being returned against the volume size.
: 1406 2396 2 !
: 1407 2397 2 !
: 1408 2398 2 IF .START_LBN + .BLOCK_COUNT GTRU .CURRENT_UCB[UCBS$L_MAXBLOCK]
: 1409 2399 2 THEN BUG_CHECK (EXTCACHIV, FATAL, 'Contents of extent cache is garbage');
: 1410 2400 2
: 1411 2401 2 ! Divide down by the volume cluster factor to convert blocks to storage
: 1412 2402 2 ! map bits. If there are non-zero remainders, reject the operation on grounds
: 1413 2403 2 ! of a bad file header.
: 1414 2404 2 !
: 1415 2405 2 !
: 1416 2406 2 IF .START_LBN MOD .CURRENT_VCB[VCBS$W_CLUSTER] NEQ 0
: 1417 2407 2 THEN BUG_CHECK (EXTCACHIV, FATAL, 'Contents of extent cache is garbage');
: 1418 2408 2 START_BIT = .START_LBN / .CURRENT_VCB[VCBS$W_CLUSTER];

```



```

: 1419      2409      2      IF .BLOCK_COUNT MOD .CURRENT_VCB[VCBSW_CLUSTER] NEQ 0
: 1420      2410      2      THEN BUG_CHECK (EXTCACHIV, FATAL, 'Contents of extent cache is garbage');
: 1421      2411      2      BIT_COUNT = .BLOCK_COUNT / .CURRENT_VCB[VCBSW_CLUSTER];
: 1422      2412      2
: 1423      2413      2
: 1424      2414      2      ! Call the bit scanner to set the appropriate
: 1425      2415      2      ! bits. Finally update the volume free block count.
: 1426      2416      2
: 1427      2417      2
: 1428      2418      2      BITSCAN (SET_BITS, .START_BIT, .BIT_COUNT, DUMMY1, DUMMY2);
: 1429      2419      2
: 1430      2420      1      END;

```

! end of routine RETURN\_BITMAP

.EXTRN BUG\$\_EXTCACHIV

				0000 0000 RETURN_BITMAP:					
				08	C2	00002	.WORD	Save nothing	: 2352
				08	AC	C1 00005	SUBL2	#8, SP	
	51	04	5E	94	AA	D0 0000B	ADDL3	BLOCK_COUNT, START_LBN, R1	: 2398
		00B0	AC		51	D1 0000F	MOVL	-108(BASE), R0	
			50		04	1B 00014	CML	R1, 176(R0)	
			50			FEFF 00016	BLEQU	1\$	
			50			0000* 00018	BUGW		: 2399
			50	98	AA	D0 0001A	.WORD	<BUG\$_EXTCACHIV!4>	
			50	3C	A0	3C 0001E	MOVL	-104(BASE), R0	: 2406
7E	00	04	AC		01	7A 00022	MOVZWL	60(R0), R0	
50	50		8E		50	7B 00028	EMUL	#1, START_LBN, #0, -(SP)	
					50	D5 0002D	EDIV	R0, (SP)+, R0, R0	
					04	13 0002F	TSTL	R0	
						FEFF 00031	BEQL	2\$	
						0000* 00033	BUGW		: 2407
			50	98	AA	D0 00035	.WORD	<BUG\$_EXTCACHIV!4>	
			51	3C	A0	3C 00039	MOVL	-104(BASE), R0	: 2408
	51	04	AC		51	C7 0003D	MOVZWL	60(R0), START_BIT	
			50	98	AA	D0 00042	DIVL3	START_BIT, START_LBN, START_BIT	
			50	3C	A0	3C 00046	MOVL	-104(BASE), R0	: 2410
7E	00	08	AC		01	7A 0004A	MOVZWL	60(R0), R0	
50	50		8E		50	7B 00050	EMUL	#1, BLOCK_COUNT, #0, -(SP)	
					50	D5 00055	EDIV	R0, (SP)+, R0, R0	
					04	13 00057	TSTL	R0	
						FEFF 00059	BEQL	3\$	
						0000* 0005B	BUGW		: 2411
			50	98	AA	D0 0005D	.WORD	<BUG\$_EXTCACHIV!4>	
			50	3C	A0	3C 00061	MOVL	-104(BASE), R0	: 2412
	50	08	AC		50	C7 00065	MOVZWL	60(R0), BIT_COUNT	
					5E	DD 0006A	DIVL3	BIT_COUNT, BLOCK_COUNT, BIT_COUNT	
				08	AE	9F 0006C	PUSHL	SP	: 2418
					50	DD 0006F	PUSHAB	DUMMY1	
					51	DD 00071	PUSHL	BIT_COUNT	
					02	DD 00073	PUSHL	START_BIT	
					05	FB 00075	PUSHL	#2	
	0000V	CF			04	0007A	CALLS	#5, BITSCAN	
							RET		: 2420

; Routine Size: 123 bytes, Routine Base: \$CODE\$ + 0974



SMALOC  
V04-001

C 10  
8-Jan-1985 18:35:14  
2-Oct-1984 12:43:38

VAX-11 Bliss-32 V4.0-742  
[F11X.BUGSRC]SMALOC.B32;1

Page 44  
(10)



```

: 1432      2421 1 ROUTINE BITSCAN (MODE, STARTBIT, BITCOUNT, STOPBIT, LENGTHFOUND) : L_NORM =
: 1433      2422 1
: 1434      2423 1 !++
: 1435      2424 1
: 1436      2425 1 FUNCTIONAL DESCRIPTION:
: 1437      2426 1
: 1438      2427 1     This routine is the basic bitmap scanner. It scans the bitmap
: 1439      2428 1     over the specified number of bits, performing the operation
: 1440      2429 1     specified by the mode.
: 1441      2430 1
: 1442      2431 1 CALLING SEQUENCE:
: 1443      2432 1     BITSCAN (ARG1, ARG2, ARG3, ARG4, ARG5)
: 1444      2433 1
: 1445      2434 1 INPUT PARAMETERS:
: 1446      2435 1     ARG1: mode of operation - see module preface
: 1447      2436 1     ARG2: starting bit address in bitmap
: 1448      2437 1     ARG3: maximum number of bits to process
: 1449      2438 1
: 1450      2439 1 IMPLICIT INPUTS:
: 1451      2440 1     CURRENT_VCB: address of VCB in process
: 1452      2441 1
: 1453      2442 1 OUTPUT PARAMETERS:
: 1454      2443 1     ARG4: address of longword to receive ending bit address
: 1455      2444 1     ARG5: address of longword to receive number of bits scanned
: 1456      2445 1
: 1457      2446 1 IMPLICIT OUTPUTS:
: 1458      2447 1     NONE
: 1459      2448 1
: 1460      2449 1 ROUTINE VALUE:
: 1461      2450 1     1 if maximum bit count processed
: 1462      2451 1     0 if not
: 1463      2452 1
: 1464      2453 1 SIDE EFFECTS:
: 1465      2454 1     bitmap blocks may be altered, read, and written
: 1466      2455 1
: 1467      2456 1 !--
: 1468      2457 1
: 1469      2458 2 BEGIN
: 1470      2459 2
: 1471      2460 2 LOCAL
: 1472      2461 2     COUNT,           ! number of bits to go
: 1473      2462 2     BLOCK,           ! current bitmap block number
: 1474      2463 2     CBYTE,           ! current byte offset in block
: 1475      2464 2     CBIT,           ! current bit number within byte
: 1476      2465 2     BYTELIM,        ! number of bytes to scan
: 1477      2466 2     BITLIM,        ! number of bits to scan
: 1478      2467 2     BUFFER,        ! address of bitmap buffer
: 1479      2468 2     ENDBYTE,       ! end of current byte scan
: 1480      2469 2     ENDBIT;       ! end of current bit scan
: 1481      2470 2
: 1482      2471 2 BIND_COMMON;
: 1483      2472 2
: 1484      2473 2 EXTERNAL ROUTINE
: 1485      2474 2     MARK_DIRTY      : L_NORM,      ! mark buffer for writeback
: 1486      2475 2     READ_BLOCK      : L_NORM;      ! read a disk block
: 1487      2476 2
: 1488      2477 2

```

```

: 1489      2478 2  ! Initialize by setting the count and setting up the pointers to
: 1490      2479 2  ! the starting position. Read the first map block. The case of a
: 1491      2480 2  ! zero count is handled specially to avoid bitmap edge problems.
: 1492      2481 2  !
: 1493      2482 2  !
: 1494      2483 2  COUNT = .BITCOUNT;
: 1495      2484 2  IF .COUNT EQL 0
: 1496      2485 2  THEN
: 1497      2486 2  BEGIN
: 1498      2487 2  .LENGTHFOUND = 0;
: 1499      2488 2  .STOPBIT = .STARTBIT;
: 1500      2489 2  RETURN 1;
: 1501      2490 2  END;
: 1502      2491 2  !
: 1503      2492 2  BLOCK = .STARTBIT<12,20>;
: 1504      2493 2  IF .BLOCK GEQU .CURRENT_VCB[VCBSB_SBMAPSIZ]
: 1505      2494 2  THEN BUG_CHECK (BADSBMB[K, FATAL, 'ACP tried to reference off end of bitmap']);
: 1506      2495 2  !
: 1507      2496 2  IF .BLOCK+1 EQL .BITMAP_VBN
: 1508      2497 2  AND .CURRENT_RVN EQL .BITMAP_RVN
: 1509      2498 2  THEN
: 1510      2499 2  BUFFER = .BITMAP_BUFFER
: 1511      2500 2  ELSE
: 1512      2501 2  BEGIN
: 1513      2502 2  BITMAP_VBN = 0;
: 1514      2503 2  BUFFER = READ_BLOCK (.BLOCK+.CURRENT_VCB[VCBSL_SBMAPLBN], 1, BITMAP_TYPE);
: 1515      2504 2  BITMAP_VBN = .BLOCK+1;
: 1516      2505 2  BITMAP_RVN = .CURRENT_RVN;
: 1517      2506 2  BITMAP_BUFFER = .BUFFER;
: 1518      2507 2  END;
: 1519      2508 2  !
: 1520      2509 2  CBYTE = .BUFFER + .STARTBIT<3,9>;
: 1521      2510 2  CBIT = .STARTBIT<0,3>;
: 1522      2511 2  !
: 1523      2512 2  ! The outer loop allows us to use the same set of bit processing instructions
: 1524      2513 2  ! for the odd bits at both the start and end of the scan.
: 1525      2514 2  !
: 1526      2515 2  !
: 1527      2516 2  WHILE 1 DO
: 1528      2517 2  BEGIN
: 1529      2518 2  !
: 1530      2519 2  ! Process bits from the starting position up to the first byte boundary.
: 1531      2520 2  !
: 1532      2521 2  !
: 1533      2522 2  BITLIM = MIN (8 - .CBIT, .COUNT); ! max number of bits to scan
: 1534      2523 2  CASE .MODE FROM 0 TO 3 OF
: 1535      2524 2  SET
: 1536      2525 2  [FIND_SET]: FFS (CBIT, BITLIM, .CBYTE, ENDBIT);
: 1537      2526 2  !
: 1538      2527 2  [FIND_CLEAR]: FFC (CBIT, BITLIM, .CBYTE, ENDBIT);
: 1539      2528 2  !
: 1540      2529 2  [SET_BITS]: BEGIN
: 1541      2530 2  (.CBYTE)<.CBIT, .BITLIM> = -1;
: 1542      2531 2  ENDBIT = .CBIT + .BITLIM;
: 1543      2532 2  END;
: 1544      2533 2  !
: 1545      2534 2  [CLEAR_BITS]: BEGIN

```



```
: 1546      2535      4      (.CBYTE)<.CBIT, .BITLIM> = 0;
: 1547      2536      4      ENDBIT = .CBIT + .BITLIM;
: 1548      2537      4      END;
: 1549      2538
: 1550      2539      TES;
: 1551      2540
: 1552      2541      ! Update the counters and pointers.
: 1553      2542      !
: 1554      2543
: 1555      2544      COUNT = .COUNT - (.ENDBIT - .CBIT);
: 1556      2545
: 1557      2546      ! If we are now positioned on a byte boundary, we can process the bitmap
: 1558      2547      on a byte by byte basis. Page through the bitmap until the count runs out.
: 1559      2548      !
: 1560      2549
: 1561      2550      IF .COUNT EQL 0 OR .ENDBIT NEQ 8 THEN EXITLOOP;
: 1562      2551
: 1563      2552      CBYTE = .CBYTE + 1;
: 1564      2553      CBIT = 0;
: 1565      2554
: 1566      2555      WHILE 1 DO
: 1567      2556      BEGIN
: 1568      2557      BYTELIM = MIN (.COUNT/8, 512 - (.CBYTE-.BUFFER));
: 1569      2558
: 1570      2559      CASE .MODE FROM 0 TO 3 OF
: 1571      2560      SET
: 1572      2561
: 1573      2562      [FIND_SET]:      ENDBYTE = CH$FIND_NOT_CH (.BYTELIM, .CBYTE, 0);
: 1574      2563
: 1575      2564      [FIND_CLEAR]:  ENDBYTE = CH$FIND_NOT_CH (.BYTELIM, .CBYTE, 255);
: 1576      2565
: 1577      2566      [SET_BITS]:      ENDBYTE = CH$FILL (255, .BYTELIM, .CBYTE);
: 1578      2567
: 1579      2568      [CLEAR_BITS]:   ENDBYTE = CH$FILL (0, .BYTELIM, .CBYTE);
: 1580      2569
: 1581      2570      TES;
: 1582      2571
: 1583      2572      IF CH$FAIL (.ENDBYTE) THEN ENDBYTE = .CBYTE + .BYTELIM;
: 1584      2573
: 1585      2574      ! If the count runs out or we run into an end condition leave the loop.
: 1586      2575      ! Otherwise read the next block, wrapping around the end of the bitmap
: 1587      2576      ! when necessary, and loop.
: 1588      2577      !
: 1589      2578
: 1590      2579      COUNT = .COUNT - (.ENDBYTE - .CBYTE) * 8;
: 1591      2580      IF .ENDBYTE - .BUFFER NEQ 512 OR .COUNT EQL 0 THEN EXITLOOP;
: 1592      2581
: 1593      2582      CASE .MODE FROM MINU (SET_BITS, CLEAR_BITS) TO MAXU (SET_BITS, CLEAR_BITS) OF
: 1594      2583      SET
: 1595      2584
: 1596      2585      [SET_BITS, CLEAR_BITS]: MARK_DIRTY (.BUFFER);
: 1597      2586
: 1598      2587      [INRANGE, OVRANGE]: 0;
: 1599      2588
: 1600      2589      TES;
: 1601      2590
: 1602      2591      BLOCK = .BLOCK + 1;
```

```

: 1603      2592  4      IF .BLOCK GEQU .CURRENT_VCB[VCBSB_SBMAPSIZ]
: 1604      2593  4      THEN BUG_CHECK (BADSBMBLK, FATAL, 'ACP tried to reference off end of bitmap');
: 1605      2594  4
: 1606      2595  4      BITMAP_VBN = 0;
: 1607      2596  4      BUFFER = READ_BLOCK (.BLOCK+.CURRENT_VCB[VCBSL_SBMAPLBN], 1, BITMAP_TYPE);
: 1608      2597  4      BITMAP_VBN = .BLOCK+1;
: 1609      2598  4      BITMAP_BUFFER = .BUFFER;
: 1610      2599  4      CBYTE = .BUFFER;
: 1611      2600  4      END;
: 1612      2601  4      ! end of block scan loop
: 1613      2602  4      ! We have either found the desired end condition or the count will run
: 1614      2603  4      ! out within the next byte. Process the final byte bit by bit.
: 1615      2604  4
: 1616      2605  4
: 1617      2606  4      IF .COUNT EQL 0 THEN EXITLOOP;
: 1618      2607  4      CBYTE = .ENDBYTE;
: 1619      2608  4      END;
: 1620      2609  4      ! end of major loop
: 1621      2610  4      ! Scan is completed. Mark the buffer dirty if necessary and return the
: 1622      2611  4      ! output values.
: 1623      2612  4
: 1624      2613  4
: 1625      2614  4      CASE .MODE FROM MINU (SET_BITS, CLEAR_BITS) TO MAXU (SET_BITS, CLEAR_BITS) OF
: 1626      2615  4      SET
: 1627      2616  4
: 1628      2617  4      [SET_BITS, CLEAR_BITS]: MARK_DIRTY (.BUFFER);
: 1629      2618  4
: 1630      2619  4      [INRANGE, OVRANGE]: 0;
: 1631      2620  4
: 1632      2621  4      TES;
: 1633      2622  4
: 1634      2623  4      .LENGTHFOUND = .BITCOUNT - .COUNT;
: 1635      2624  4      .STOPBIT = .STARTBIT + ..LENGTHFOUND;
: 1636      2625  4      RETURN .COUNT EQL 0;
: 1637      2626  4
: 1638      2627  1      END;
! end of routine BITSCAN

```

				.EXTRN MARK_DIRTY, READ_BLOCK				
				.EXTRN BUGS_BADSBMBLK				
				OBFC	0000	BITSCAN: .WORD	Save R2,R3,R4,R5,R6,R7,R8,R9,R11	2421
	5E			OC	C2	SUBL2	#12, SP	2460
			B4	AA	9F	PUSHAB	-76(BASE)	2483
	59			OC	D0	MOVL	BITCOUNT, COUNT	2484
				OC	12	BNEQ	1\$	2487
			14	BC	D4	CLRL	@LENGTHFOUND	2488
	10	BC		AC	D0	MOVL	STARTBIT, @STOPBIT	2489
		50		01	D0	MOVL	#1, R0	2492
					04	RET		2493
57	09	AC		04	EF	EXTZV	#4, #20, STARTBIT+1, BLOCK	2494
			14	AA	D0	MOVL	-104(BASE), R0	
			50	00	ED	CMPZV	#0, #8, 57(R0), BLOCK	
57	39	A0		04	1A	BGTRU	2\$	
			08		FEFF	BUGW		2494
					0000*	.WORD	<BUGS_BADSBMBLK!4>	
					0002E			





	66	OC	AE		00	3B	000FE	19\$:	SKPC	24\$-18\$		2562
					0A	13	00103		BEQL	#0, BYTE LIM, (C BYTE)		
	66	OC	AE	FF	0A	11	00105		BRB	21\$		
					8F	3B	00107	20\$:	SKPC	22\$		2564
					02	12	0010D		BNEQ	#255, BYTE LIM, (C BYTE)		
					51	D4	0010F	21\$:	CLRL	R1		
					51	D0	00111	22\$:	MOVL	R1, ENDBYTE		
OC	AE	FF	8F		14	11	00114		BRB	26\$		
					00	2C	00116	23\$:	MOVCS	#0, (SP), #255, BYTE LIM, (C BYTE)		2566
					66		0011D					
OC	AE				07	11	0011E		BRB	25\$		
					00	2C	00120	24\$:	MOVCS	#0, (SP), #0, BYTE LIM, (C BYTE)		2568
					66		00126					
					52		53 D0 00127	25\$:	MOVL	R3, ENDBYTE		
	52		56	OC	05	12	0012A	26\$:	BNEQ	27\$		2572
	50		56		AE	C1	0012C		ADDL3	BYTE LIM, C BYTE, ENDBYTE		
			56		52	C3	00131	27\$:	SUBL3	ENDBYTE, C BYTE, RO		2579
			59		6940	7E	00135		MOVAQ	(COUNT)[RO], COUNT		
	50	08	AE	00000200	8F	C1	00139		ADDL3	#512, BUFFER, RO		2580
			50		52	D1	00142		CPL	ENDBYTE, RO		
					52	12	00145		BNEQ	32\$		
					59	D5	00147		TSTL	COUNT		
					4E	13	00149		BEQL	32\$		
	01		02		04	AC	0014B		CASEL	MODE, #2, #1		2582
			0006		0006		00150	28\$:	.WORD	29\$-28\$,- 29\$-28\$		
					08	11	00154		BRB	30\$		
					08	AE	DD 00156	29\$:	PUSHL	BUFFER		2585
		0000G	CF		01	FB	00159		CALLS	#1, MARK_DIRTY		
					57	D6	0015E	30\$:	INCL	BLOCK		2591
			50		98	AA	DD 00160		MOVL	-104(BASE), RO		2592
	57	39	AO		00	ED	00164		CMPZV	#0, #8, 57(RO), BLOCK		
			08		04	1A	0016A		BGTRU	31\$		
							FEFF 0016C		BUGW			2593
							0000* 0016E		.WORD	<BUG\$ BADSBMBLK!4>		
					00	BE	D4 00170	31\$:	CLRL	@0(SPT)		2595
					01	DD	00173		PUSHL	#1		2596
					01	DD	00175		PUSHL	#1		
			50		98	AA	DD 00177		MOVL	-104(BASE), RO		
		0000G	CF		34	B047	9F 0017B		PUSHAB	@52(RO)[BLOCK]		
		08	AE			03	FB 0017F		CALLS	#3, READ_BLOCK		
		00	BE		01	A7	D0 00184		MOVL	RO, BUFFER		2597
		BC	AA		08	AE	D0 0018D		MOVAB	1(R7), @0(SP)		2598
			56		08	AE	D0 00192		MOVL	BUFFER, -68(BASE)		2599
					FF3E	31	00196		BRW	BUFFER, CBYTE		2555
					59	D5	00199	32\$:	TSTL	COUNT		2606
					06	13	0019B		BEQL	33\$		
			56		52	D0	0019D		MOVL	ENDBYTE, CBYTE		2607
					FEDC	31	001A0		BRW	5\$		2516
			02		04	AC	CF 001A3	33\$:	CASEL	MODE, #2, #1		2614
			0006		0006		001A8	34\$:	.WORD	35\$-34\$,- 35\$-34\$		
					08	11	001AC		BRB	36\$		
		0000G	CF		08	AE	DD 001AE	35\$:	PUSHL	BUFFER		2617
					01	FB	001B1		CALLS	#1, MARK_DIRTY		



```

14 BC      OC AC      59 C3 001B6 36$:  SUBL3  COUNT, BITCOUNT, @LENGTHFOUND
10 BC      08 AC      14 BC C1 001BC  ADDL3  @LENGTHFOUND, STARTBIT, @STOPBIT
                    50 D4 001C3  CLRL   RO
                    59 D5 001C5  TSTL  COUNT
                    02 12 001C7  BNEQ  37$
                    50 D6 001C9  INCL  RO
                    04 001CB 37$:  RET

```

```

: 2623
: 2624
: 2625
:
: 2627

```

: Routine Size: 460 bytes, Routine Base: \$CODE\$ + 09EF

```

: 1639      2628 1
: 1640      2629 1 END
: 1641      2630 0 ELUDOM

```

PSECT SUMMARY

Name	Bytes	Attributes
\$CODE\$	3003	NOVEC, NOWRT, RD, EXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2)

Library Statistics

File	Symbols		Pages Mapped	Processing Time
	Total	Loaded Percent		
_\$255\$DUA18:[SYSLIB]LIB.L32;1	18619	59 0	1000	00:01.9

COMMAND QUALIFIERS

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

```

: Size:      2971 code + 32 data bytes
: Run Time:   02:12.6
: Elapsed Time: 03:12.9
: Lines/CPU Min: 1190
: Lexemes/CPU-Min: 53134
: Memory Used: 339 pages
: Compilation Complete

```



