


```

1 0001 0 MODULE VMOUNT (
2 0002 0
3 0003 0     LANGUAGE (BLISS32),
4 0004 0     ADDRESSING MODE (NONEXTERNAL = LONG_RELATIVE),
5 0005 0     IDENT = 'V04-002'
6 0006 1 BEGIN
7 0007 1
8 0008 1
9 0009 1
10 0010 1
11 0011 1 * *****
12 0012 1 *  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
13 0013 1 *  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
14 0014 1 *  ALL RIGHTS RESERVED.
15 0015 1 *
16 0016 1 *  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
17 0017 1 *  ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
18 0018 1 *  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
19 0019 1 *  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
20 0020 1 *  OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
21 0021 1 *  TRANSFERRED.
22 0022 1 *
23 0023 1 *  THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
24 0024 1 *  AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
25 0025 1 *  CORPORATION.
26 0026 1 *
27 0027 1 *  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
28 0028 1 *  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
29 0029 1 *
30 0030 1 * *****
31 0031 1
32 0032 1 **
33 0033 1
34 0034 1 FACILITY: MOUNT Utility Structure Levels 1 & 2
35 0035 1
36 0036 1 ABSTRACT:
37 0037 1
38 0038 1     This is the main routine of the MOUNT utility. It provides the
39 0039 1     general control flow of the MOUNT command and contains most of
40 0040 1     the base data structures.
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: 5-Sep-1977 16:58
51 0051 1
52 0052 1 MODIFIED BY:
53 0053 1
54 0054 1     V04-002 HH0056 Hai Huang 10-Sep-1984
55 0055 1     Suppress outputting VOLINV error messages during
56 0056 1     VOLINV retries.
57 0057 1

```

| | | | | | | |
|-----|------|---|---------|---------|----------------------|--|
| 58 | 0058 | 1 | V04-001 | HM0055 | Hai Huang | 06-Sep-1984 |
| 59 | 0059 | 1 | | | | Send mount/cluster requests with operator assist disabled. |
| 60 | 0060 | 1 | | | | |
| 61 | 0061 | 1 | | | | |
| 62 | 0062 | 1 | V03-035 | CDS0005 | Christian D. Saether | 29-Aug-1984 |
| 63 | 0063 | 1 | | | | Call STAND ALONE REBUILD routine which will only do rebuild if necessary at that time. |
| 64 | 0064 | 1 | | | | |
| 65 | 0065 | 1 | | | | |
| 66 | 0066 | 1 | V03-034 | HM0043 | Hai Huang | 07-Aug-1984 |
| 67 | 0067 | 1 | | | | Wait a while before retrying IOC\$SEARCH. |
| 68 | 0068 | 1 | | | | |
| 69 | 0069 | 1 | V03-033 | HM0042 | Hai Huang | 27-Jul-1984 |
| 70 | 0070 | 1 | | | | Clear the global lock storage area during run time. |
| 71 | 0071 | 1 | | | | |
| 72 | 0072 | 1 | V03-032 | HM0041 | Hai Huang | 24-Jul-1984 |
| 73 | 0073 | 1 | | | | Remove REQUIRE 'LIBDS:[VMSLIB.OBJ]MOUNTMSG.B32'. |
| 74 | 0074 | 1 | | | | |
| 75 | 0075 | 1 | V03-031 | HM0037 | Hai Huang | 12-Jul-1984 |
| 76 | 0076 | 1 | | | | Make the label lock node-specific, i.e. make the CSID part of the label lock. |
| 77 | 0077 | 1 | | | | |
| 78 | 0078 | 1 | | | | |
| 79 | 0079 | 1 | V03-030 | HM0036 | Hai Huang | 11-Jul-1984 |
| 80 | 0080 | 1 | | | | Send the mount request cluster wide even if the volume is already mounted on the local node. |
| 81 | 0081 | 1 | | | | |
| 82 | 0082 | 1 | | | | |
| 83 | 0083 | 1 | V03-029 | HM0034 | Hai Huang | 09-Jul-1984 |
| 84 | 0084 | 1 | | | | Add yet another interlock to serialize shared mounts. |
| 85 | 0085 | 1 | | | | |
| 86 | 0086 | 1 | V03-028 | HM0032 | Hai Huang | 05-Jul-1984 |
| 87 | 0087 | 1 | | | | For private mounts, transfer device ownership to the top level process in the process tree. |
| 88 | 0088 | 1 | | | | |
| 89 | 0089 | 1 | | | | |
| 90 | 0090 | 1 | V03-027 | HM0024 | Hai Huang | 18-Jun-1984 |
| 91 | 0091 | 1 | | | | Do not call IOC\$LOCK_DEV to test mode of the device lock, as this routine could corrupt the lock value block. |
| 92 | 0092 | 1 | | | | |
| 93 | 0093 | 1 | | | | |
| 94 | 0094 | 1 | V03-026 | HM0021 | Hai Huang | 14-May-1984 |
| 95 | 0095 | 1 | | | | Refine HM0019 to mark the device as allocated after IOC\$SEARCH while holding the I/O database mutex. Also, reject private mounts if IOC\$SEARCH failed. |
| 96 | 0096 | 1 | | | | |
| 97 | 0097 | 1 | | | | |
| 98 | 0098 | 1 | | | | |
| 99 | 0099 | 1 | V03-025 | HM0019 | Hai Huang | 07-May-1984 |
| 100 | 0100 | 1 | | | | Properly interlock simultaneous mounts in a cluster-environment. |
| 101 | 0101 | 1 | | | | |
| 102 | 0102 | 1 | | | | |
| 103 | 0103 | 1 | V03-024 | HM0016 | Hai Huang | 23-Apr-1984 |
| 104 | 0104 | 1 | | | | Get the device name if IOC\$SEARCH failed. |
| 105 | 0105 | 1 | | | | |
| 106 | 0106 | 1 | V03-023 | HM0015 | Hai Huang | 20-Apr-1984 |
| 107 | 0107 | 1 | | | | Get IOC\$SEARCH to return the lock value block of the device lock. |
| 108 | 0108 | 1 | | | | |
| 109 | 0109 | 1 | | | | |
| 110 | 0110 | 1 | V03-022 | HM0010 | Hai Huang | 30-Mar-1984 |
| 111 | 0111 | 1 | | | | Fix generic mount. |
| 112 | 0112 | 1 | | | | |
| 113 | 0113 | 1 | V03-021 | HM0004 | Hai Huang | 09-Mar-1984 |
| 114 | 0114 | 1 | | | | Add cluster-wide mount support. |

| | | | | | |
|-----|------|---|---------|---|------------------|
| 115 | 0115 | 1 | | | |
| 116 | 0116 | 1 | V03-020 | HM0002 Hai Huang | 02-Feb-1984 |
| 117 | 0117 | 1 | | Add job-wide mount support. | |
| 118 | 0118 | 1 | | | |
| 119 | 0119 | 1 | V03-019 | ACG0369 Andrew C. Goldstein, | 8-Nov-1983 11:24 |
| 120 | 0120 | 1 | | Don't issue IOS_AVAILABLE on mount failure of mounted disk | |
| 121 | 0121 | 1 | | | |
| 122 | 0122 | 1 | V03-018 | CDS0004 Christian D. Saether | 13-Sep-1983 |
| 123 | 0123 | 1 | | Only clear VALID for tapes in the main error handler. | |
| 124 | 0124 | 1 | | Move the CLEAR_VALID routine here from RDHOME as it | |
| 125 | 0125 | 1 | | is only referenced here now. | |
| 126 | 0126 | 1 | | | |
| 127 | 0127 | 1 | V03-017 | TCM0003 Trudy C. Matthews | 07-Sep-1983 |
| 128 | 0128 | 1 | | When converting the exclusive device lock to a shared lock, | |
| 129 | 0129 | 1 | | make sure it is still system-owned. | |
| 130 | 0130 | 1 | | | |
| 131 | 0131 | 1 | V03-016 | TCM0002 Trudy C. Matthews | 01-Sep-1983 |
| 132 | 0132 | 1 | | Make allocating a device followed by mounting a shared | |
| 133 | 0133 | 1 | | volume on that device work correctly (i.e. deallocate | |
| 134 | 0134 | 1 | | the device and convert the lock to CR mode). | |
| 135 | 0135 | 1 | | | |
| 136 | 0136 | 1 | V03-015 | CDS0003 Christian D. Saether | 5-Aug-1983 |
| 137 | 0137 | 1 | | Add cluster consistency checking routines. | |
| 138 | 0138 | 1 | | Add status block to GETDVIW call so that wait | |
| 139 | 0139 | 1 | | always works correctly. | |
| 140 | 0140 | 1 | | | |
| 141 | 0141 | 1 | V03-014 | CDS0002 Christian D. Saether | 3-Aug-1983 |
| 142 | 0142 | 1 | | Remove the device ref count check prior to assigning | |
| 143 | 0143 | 1 | | the channel (from tcm0001) as it was racy. | |
| 144 | 0144 | 1 | | | |
| 145 | 0145 | 1 | V03-013 | STJ3015 Steven T. Jeffreys | 30-Jul-1983 |
| 146 | 0146 | 1 | | Fix link-time truncation error. | |
| 147 | 0147 | 1 | | | |
| 148 | 0148 | 1 | V03-012 | TCM0001 Trudy C. Matthews | 28-Jul-1983 |
| 149 | 0149 | 1 | | Re-write the MOUNT_VOLUME routine so that it uses a | |
| 150 | 0150 | 1 | | mount interlock rather than temporarily allocating the | |
| 151 | 0151 | 1 | | volume. Also ensure that cluster-wide locks are taken | |
| 152 | 0152 | 1 | | out in the appropriate mode (EX for private mounts and | |
| 153 | 0153 | 1 | | CR for shared mounts). | |
| 154 | 0154 | 1 | | | |
| 155 | 0155 | 1 | V03-011 | STJ3113 Steven T. Jeffreys, | 26-Jul-1983 |
| 156 | 0156 | 1 | | Moved ACTIVATE_JOURNAL and helper routines to their own | |
| 157 | 0157 | 1 | | module, RUJMAN. | |
| 158 | 0158 | 1 | | | |
| 159 | 0159 | 1 | V03-010 | STJ3111 Steven T. Jeffreys, | 18-Jul-1983 |
| 160 | 0160 | 1 | | When the privileges are amplified, take pains to include | |
| 161 | 0161 | 1 | | those privileges that are in the second longword of the | |
| 162 | 0162 | 1 | | privilege mask, notably PRMJNL privilege. | |
| 163 | 0163 | 1 | | | |
| 164 | 0164 | 1 | V03-009 | DMW4045 DMWalp | 7-Jun-1983 |
| 165 | 0165 | 1 | | Remove (S)LOG_Entry | |
| 166 | 0166 | 1 | | | |
| 167 | 0167 | 1 | V03-008 | CDS0001 Christian D. Saether | 28-May-1983 |
| 168 | 0168 | 1 | | Tolerate allocation failure for F11E mounts. | |
| 169 | 0169 | 1 | | | |
| 170 | 0170 | 1 | V03-007 | STJ3102 Steven T. Jeffreys, | 25-May-1983 |
| 171 | 0171 | 1 | | - Add call to \$CREJNL. | |

```

172 0172 1
173 0173 1
174 0174 1
175 0175 1
176 0176 1
177 0177 1
178 0178 1
179 0179 1
180 0180 1
181 0181 1
182 0182 1
183 0183 1
184 0184 1
185 0185 1
186 0186 1
187 0187 1
188 0188 1
189 0189 1
190 0190 1
191 0191 1
192 0192 1
193 0193 1
194 0194 1
195 0195 1
196 0196 1
197 0197 1
198 0198 1
199 0199 1
200 0200 1
201 0201 1
202 0202 1
203 0203 1
204 0204 1
205 0205 1
206 0206 1
207 0207 1
208 0208 1
209 0209 1
210 0210 1
211 0211 1
212 0212 1
213 0213 1
214 0214 1
215 0215 1
216 0216 1
217 0217 1
218 0218 1
219 0219 1
220 0220 1
221 0221 1
222 0222 1
223 0223 1
224 0224 1
225 0225 1
226 0226 1
227 0227 1
228 0228 1

```

```

V03-006 MMD0115 Meg Dumont, 29-Mar-1983 0:39
Add OPT_OVR_VOLO to override options set

V03-005 STJ3061 Steven T. Jeffreys, 08-Mar-1983
- Grant user PSWAPM privilege. Needed to create ACP.

V03-004 STJ50311 Steven T. Jeffreys, 11-Feb-1983
- Make all uses of PHYS_NAME indexed by DEVICE_INDEX
- Ensure DEVICE_INDEX is not reset on retry
- Remove references to FIRST_CHANNEL.
- Make CALLERS_ACMOD a global cell containing the
caller's access mode.
- Changed device allocation/deallocation logic.
Moved routine DEALLOCATE_DEVICE to ASSIST.

V03-003 STJ3037 Steven T. Jeffreys, 14-Oct-1982
If the mount attempt fails, free up the drive(s) via
an IOS_AVAILABLE $qio.

V03-002 KTA0103 Kerbey T. Altmann 29-Jun-1982
Change a register to NOPRESERVE in DEALLOCATE_DEVICE.

V03-001 STJ0252 Steven T. Jeffreys, 03-Apr-1982
- Allocate devices in the access mode of the caller.
- Check allocation return status and terminate the mount
attempt if the specified device does not exist.
- Manually deallocate shared disk volumes after
they are mounted. This is necessitated by a
change to $DALLOC such that mounted volumes may
no longer be deallocated.

V02-020 STJ0229 Steven T. Jeffreys, 01-Mar-1982
- Set inhibit message bit in the exit status code
if the message text was written via $PUTMSG.

V02-019 STJ0190 Steven T. Jeffreys, 02-Feb-1982
- Zero OWN and GLOBAL storage to guaranty restartability.

V02-018 STJ0170 Steven T. Jeffreys, 13-Jan-1982
More work for $MOUNT support.

V02-017 RNG0001 Rod N. Gamache 05-Jan-1982
Declare MOUNT_OPTIONS to be external.

V02-016 STJ0161 Steven T. Jeffreys, 04-Jan-1982
Changed OPT_OVERLOCK to OPT_OVR_LOCK. Do not print
messages if OPT_MESSAGE is not set.

V02-015 ACG0246 Andrew C. Goldstein, 4-Jan-1982 15:21
Add /OVER:LOCK

V02-014 STJ0149 Steven T. Jeffreys 02-Jan-1981
Extensive rewrite to support the $MOUNT system service.

V02-013 STJ0089 Steven T. Jeffreys 09-Aug-1981
Reset mount options at the beginning of each attempt

```

```

229 0229 1 to mount a volume.
230 0230 1
231 0231 1 V02-012 DMW0008 David Michael Walp 10-Jun-1981
232 0232 1 Liberal re-write to facilitate operator assisted mount
233 0233 1 work for tapes.
234 0234 1
235 0235 1 V0111 STJ0005 Steven T. Jeffreys, 9-Oct-1980
236 0236 1 Liberal re-write to facilitate operator assisted mount.
237 0237 1
238 0238 1 V0110 ACG0125 Andrew C. Goldstein, 23-Jan-1980 14:57
239 0239 1 Init USER_STATUS cell for correct header error reporting
240 0240 1
241 0241 1 V0109 ACG0123 Andrew C. Goldstein, 17-Jan-1980 20:33
242 0242 1 Complete integration of disk rebuild
243 0243 1
244 0244 1 V0108 RIH0051 Richard I. Mustvedt, 13-Jan-1979 14:33
245 0245 1 Add call to rebuild bitmaps and quota file on volume mount.
246 0246 1
247 0247 1 V0107 ACG0079 Andrew C. Goldstein, 5-Nov-1979 13:53
248 0248 1 Structures for file ID and extent cacheing
249 0249 1
250 0250 1 V0106 ACG0072 Andrew C. Goldstein, 15-Oct-1979 16:12
251 0251 1 Check primary and secondary device characteristics
252 0252 1
253 0253 1 V0105 ACG0069 Andrew C. Goldstein, 8-Oct-1979 18:32
254 0254 1 Remove device data table
255 0255 1
256 0256 1 V0104 ACG0044 Andrew C. Goldstein, 18-Jun-1979 16:15
257 0257 1 Add disk quota support
258 0258 1
259 0259 1 V0103 ACG21786 Andrew C. Goldstein, 2-Feb-1979 14:19
260 0260 1 Fix home block scan loop limit conditional
261 0261 1
262 0262 1 V0102 ACG0013 Andrew C. Goldstein, 5-Jan-1979 13:52
263 0263 1 Don't clear valid bit on failure on already mounted volume
264 0264 1
265 0265 1 V0101 ACG0003 Andrew C. Goldstein, 27-Nov-1978 17:48
266 0266 1 Add multi-volume support for disk
267 0267 1
268 0268 1 V0100 ACG00001 Andrew C. Goldstein, 10-Oct-1978 19:56
269 0269 1 Previous revision history moved to [MOUNT.SRC]MOUNT.REV
270 0270 1 **
271 0271 1
272 0272 1
273 0273 1 LIBRARY 'SYSS$LIBRARY:LIB.L32';
274 0274 1 REQUIRE 'SRCS:MOUDEF.B32';
275 0806 1 REQUIRE 'LIBDS:VMSLIB.OBJ]INITMSG.REQ';
276 0938 1
277 0939 1
278 0940 1 FORWARD ROUTINE
279 0941 1 SYSS$VMOUNT, entry point (w/o operator assist)
280 0942 1 VMOUNT ENVELOPE, base call frame for MOUNT VOLUME
281 0943 1 REBUILD ENVELOPE, base call frame for REBUILD
282 0944 1 INTERCEPT SIGNAL, Intercept EXEC mode signal
283 0945 1 MOUNT VOLUME, Mount a given volume
284 0946 1 MAIN HANDLER, main condition handler
285 0947 1 FORCE_DISMOUNT, dismount a volume just mounted

```

VMOUNT
V04-002

1 4
16-Sep-1984 01:00:56 VAX-11 Bliss-32 V4.0-742 Page 6
12-Sep-1984 11:14:53 DISK\$VMSMASTER:[MOUNT.SRC]VMOUNT.B32;3 (1)

| | | | | | |
|---|-----|------|---|-----------------|------------|
| : | 286 | 0948 | 1 | CLEAR_VALID, | |
| : | 287 | 0949 | 1 | DALLOC_SHR_DEV, | |
| : | 288 | 0950 | 1 | XFER_DEV_OWNER, | |
| : | 289 | 0951 | 1 | MOUNT_CLUSTER, | |
| : | 290 | 0952 | 1 | MOUNT_ENCIPHER, | |
| : | 291 | 0953 | 1 | SEARCH_DEVICE, | |
| : | 292 | 0954 | 1 | DEQ_MOUNT_LOCK | : NOVALUE, |
| : | 293 | 0955 | 1 | WAIT_DELTA | : NOVALUE; |
| : | 294 | 0956 | 1 | | |

| | |
|---|--|
| : | Clear VALID flag in UCB. |
| : | deallocate device for shared mount |
| : | transfer device ownership |
| : | cluster-wide mount |
| : | create a cluster-mount packet |
| : | generic device search/allocate routine |
| : | dequeue the mount lock |
| : | wait before IOC\$SEARCH retry |


```

296 0957 1 | +
297 0958 1 |
298 0959 1 | Own storage for general use in the MOUNT utility
299 0960 1 | Note that DATA_BASE_READY and STORED_CONTEXT initialized
300 0961 1 | in the module ASSIST.
301 0962 1 |
302 0963 1 | -
303 0964 1 |
304 0965 1 | GLOBAL
305 0966 1 | VMOUNT_GBL_START: VECTOR [0]; ! Mark start of global storage
306 0967 1 | STORED_CONTEXT : BITVECTOR [32]; ! store the context of some 1 time only
307 0968 1 | DATA_BASE_READY : LONG; Boolean
308 0969 1 | DEV_ALLOCATED : BITVECTOR [DEVMAX] VOLATILE; !Indicates which physical devices are allocated
309 0970 1 | DEV_ACQUIRED : BITVECTOR [DEVMAX] VOLATILE;
310 0971 1 | !Indicates which devices have been
311 0972 1 | interlocked.
312 0973 1 | LOCK_STATUS : VECTOR [2]; ! Lock status block for $ENQ.
313 0974 1 | CLEANUP_ALLOC : BITVECTOR [DEVMAX] VOLATILE; !Indicates which physical devices need to be dealloc
314 0975 1 | CLEANUP_FLAGS : BITVECTOR [32] VOLATILE; ! error cleanup status flags
315 0976 1 | CHANNEL : LONG VOLATILE; ! channel number for I/O
316 0977 1 | DEVICE_INDEX : LONG VOLATILE; ! Index into device list
317 0978 1 | MAILBOX_CHANNEL : LONG VOLATILE; ! channel number of ACP termination mailbox
318 0979 1 | CALLERS_ACMOD : LONG; ! caller's access mode
319 0980 1 | PHYS_COUNT : LONG; ! number of physical devices in use
320 0981 1 | PHYS_NAME : VECTOR [DEVMAX*2]; ! descriptor of physical device name
321 0982 1 | NAME_BUFFER : VECTOR [NAMEBUF_LEN*DEVMAX, BYTE];
322 0983 1 | ! string buffer for physical device name
323 0984 1 | LOG_BUFFER : VECTOR [63, BYTE];
324 0985 1 | ! buffer to construct logical name
325 0986 1 | HOME_BLOCK : BBLOCK [512]; ! buffer for volume header label or home block
326 0987 1 | DEVICE_CHAR : BBLOCK [DIB$K_LENGTH];
327 0988 1 | ! buffer for device characteristics
328 0989 1 | DEVICE_CHAR2 : BBLOCK [DIB$K_LENGTH];
329 0990 1 | ! buffer for sec. device characteristics
330 0991 1 | HOMEBLOCK_LBN, ! LBN of home block read
331 0992 1 | HEADER_LBN, ! LBN of file header read
332 0993 1 | CURRENT_RVN, ! RVN of disk being mounted
333 0994 1 | USER_STATUS : VECTOR [2]; ! status return for various routines
334 0995 1 | CURRENT_VCB : REF BBLOCK; ! address of VCB used by CHECK_HEADER2
335 0996 1 | REAL_MVC : REF BBLOCK; ! address of MVC allocated for mag tape volume
336 0997 1 | REAL_RVT : REF BBLOCK; ! address of RVT allocated for mag tape volume
337 0998 1 | REAL_VCB : REF BBLOCK; ! address of VCB allocated for volume
338 0999 1 | REAL_VCA : REF BBLOCK; ! address of cache block allocated for vol.
339 1000 1 | REAL_FCB : REF BBLOCK; ! address of FCB allocated for volume
340 1001 1 | REAL_WCB : REF BBLOCK; ! address of window allocated for volume
341 1002 1 | REAL_AQB : REF BBLOCK; ! address of AQB allocated for volume
342 1003 1 | MTL_ENTRY : REF BBLOCK; ! address of mounted volume list entry
343 1004 1 | SMT_ENTRY : REF BBLOCK; ! address of volume set MTL
344 1005 1 | MOUNT_ITMLST, ! address of the mount item list
345 1006 1 | LABLCK_STATUS : VECTOR [2]; ! label lock status
346 1007 1 | VMOUNT_GBL_END : VECTOR [0]; ! Mark end of GLOBAL storage
347 1008 1 |
348 1009 1 |
349 1010 1 | GLOBAL BIND
350 1011 1 | VOL1 = HOME_BLOCK; ! BUFFER FOR VOL1 MAGNETIC TAPE LABEL
351 1012 1 |
352 1013 1 | GLOBAL

```

```

: 353      1014 1      ALLDEVNAM_BUF : VECTOR [NAMEBUF_LEN, BYTE]
: 354      1015 1      INITIAL (BYTE ('MOUS',
: 355      1016 1      REP NAMEBUF_LEN-4 OF (' ')),
: 356      1017 1      ! string buffer for alloc class devnam
: 357      1018 1      ALLDEVNAM_DESC : VECTOR [2] INITIAL (0, ALLDEVNAM_BUF),
: 358      1019 1      ! descriptor for alloc class devnam
: 359      1020 1      DEVCHAR_DESC : VECTOR [2] INITIAL (DIBSK_LENGTH, DEVICE_CHAR),
: 360      1021 1      ! descriptor for device characteristics
: 361      1022 1      DEVCHAR_DESC2 : VECTOR [2] INITIAL (DIBSK_LENGTH, DEVICE_CHAR2),
: 362      1023 1      ! descriptor for sec. device characteristics
: 363      1024 1
: 364      1025 1      LABLCKNAM_BUF : VECTOR [NAMEBUF_LEN+4, BYTE]
: 365      1026 1      INITIAL (BYTE ('MOUS',
: 366      1027 1      REP NAMEBUF_LEN OF (' ')),
: 367      1028 1      ! label lock-name buffer
: 368      1029 1      LABLCKNAM_DESC : VECTOR [2, LONG]
: 369      1030 1      INITIAL (0, LABLCKNAM_BUF);
: 370      1031 1      ! label lock descriptor
: 371      1032 1

```

```

373 1033 1 GLOBAL ROUTINE SYSS$VMOUNT (ITEM_LIST) =
374 1034 1 ++
375 1035 1
376 1036 1 FUNCTIONAL DESCRIPTION:
377 1037 1
378 1038 1 This is the main routine of the MOUNT utility.
379 1039 1
380 1040 1 CALLING SEQUENCE:
381 1041 1 $MOUNT (arglist)
382 1042 1
383 1043 1 INPUT PARAMETERS:
384 1044 1 ITEM_LIST : Address of a $GETJPI-like item list
385 1045 1
386 1046 1 IMPLICIT INPUTS:
387 1047 1 NONE
388 1048 1
389 1049 1 OUTPUT PARAMETERS:
390 1050 1 NONE
391 1051 1
392 1052 1 IMPLICIT OUTPUTS:
393 1053 1 NONE
394 1054 1
395 1055 1 ROUTINE VALUE:
396 1056 1 assorted status values
397 1057 1
398 1058 1 SIDE EFFECTS:
399 1059 1 volume(s) mounted, device data base updated
400 1060 1
401 1061 1 --
402 1062 1
403 1063 2 BEGIN
404 1064 2
405 1065 2 BUILTIN
406 1066 2 MOVPSL, ! Get current PSL
407 1067 2 CALLG, ! Used to call CHECK_PARAMS
408 1068 2 AP; ! Used to pass params to CHECK_PARAMS
409 1069 2
410 1070 2 EXTERNAL ROUTINE
411 1071 2 ACTIVATE JOURNAL: ADDRESSING_MODE (GENERAL), ! activate RUJ
412 1072 2 $DALLOC DEVSSU : ADDRESSING_MODE (GENERAL),
413 1073 2 CHECK_PARAMS; ! Process the user-supplied parameters
414 1074 2
415 1075 2 EXTERNAL
416 1076 2 DEVICE_COUNT : ADDRESSING_MODE (GENERAL)
417 1077 2 LONG, ! Number of devices specified
418 1078 2 LCK_GLOBAL_START: ADDRESSING_MODE (GENERAL),
419 1079 2 ! Start of global lock area
420 1080 2 LCK_GLOBAL_END : ADDRESSING_MODE (GENERAL);
421 1081 2 ! End of global lock area
422 1082 2
423 1083 2 LOCAL
424 1084 2
425 1085 2 ! Declare the privileges that are necessary for MOUNT to work.
426 1086 2
427 1087 2 CURRENT_PSL : BBLOCK [4], ! holds current PSL
428 1088 2 MOUNT_PRIVS : BBLOCK [8], ! Amplified privilege mask
429 1089 2 USER_PRIVS : BBLOCK [8], ! Temp storage for privilege mask

```

```

: 430      1090      2      STATUS;                                ! system service status
: 431      1091      2
: 432      1092      2
: 433      1093      2      CHANNEL      = 0;
: 434      1094      2      USER_STATUS   = 1;
: 435      1095      2
: 436      1096      2      MOUNT_PRIVS  = (1^$BITPOSITION (PRV$V_ACNT)   OR      ! Amplified privilege mask
: 437      1097      2      1^$BITPOSITION (PRV$V_ALT_PRI) OR
: 438      1098      2      1^$BITPOSITION (PRV$V_BUGCHK) OR
: 439      1099      2      1^$BITPOSITION (PRV$V_BYPASS) OR
: 440      1100      2      1^$BITPOSITION (PRV$V_DETACH) OR
: 441      1101      2      1^$BITPOSITION (PRV$V_EXQUOTA) OR
: 442      1102      2      1^$BITPOSITION (PRV$V_GROUP) OR
: 443      1103      2      1^$BITPOSITION (PRV$V_MOUNT) OR
: 444      1104      2      1^$BITPOSITION (PRV$V_PHY_IO) OR
: 445      1105      2      1^$BITPOSITION (PRV$V_PSWAPM) OR
: 446      1106      2      1^$BITPOSITION (PRV$V_TMPMBX) OR
: 447      1107      2      1^$BITPOSITION (PRV$V_SETPRV) OR
: 448      1108      2      1^$BITPOSITION (PRV$V_SYSLCK) OR
: 449      1109      2      1^$BITPOSITION (PRV$V_WORLD)
: 450      1110      2      );
: 451      1111      2      MOUNT_PRIVS[PRV$V_PRMJNL] = 1;          ! PRMJNL is in the 2nd longword
: 452      1112      2
: 453      1113      2      ! Process the user-supplied parameters, if
: 454      1114      2      ! we haven't already. The conditional call
: 455      1115      2      ! is to save the overhead of having to do it
: 456      1116      2      ! for each attempt at a mount, and to make
: 457      1117      2      ! sure that it is done at least once.
: 458      1118      2
: 459      1119      2
: 460      1120      2      IF NOT .DATA_BASE_READY
: 461      1121      2      THEN
: 462      1122      2      BEGIN
: 463      1123      2      CH$FILL (0, VMOUNT_GBL_END-VMOUNT_GBL_START, VMOUNT_GBL_START);
: 464      1124      2      CH$FILL (0, LCK_GLOBAL_END-LCK_GLOBAL_START, LCK_GLOBAL_START);
: 465      1125      2      MOUNT_ITMLST = .ITEM_LIST;
: 466      1126      2      DATA_BASE_READY = 1;
: 467      1127      2      IF NOT (STATUS = CALLG (.AP, CHECK_PARAMS))
: 468      1128      2      THEN
: 469      1129      2      RETURN (.STATUS);
: 470      1130      2      MOVPSL (CURRENT_PSL);
: 471      1131      2      CALLERS_ACMOD = .CURRENT_PSL [PSL$V_PVMOD];
: 472      1132      2      END;
: 473      1133      2
: 474      1134      2      ! Save the current privilege mask and grant the
: 475      1135      2      ! caller the necessary privileges.
: 476      1136      2
: 477      1137      2
: 478      1138      2      $SETPRV (ENBFLG=1, PRVADR=MOUNT_PRIVS, PRVPRV=USER_PRIVS);
: 479      1139      2
: 480      1140      2      ! Loop for all devices in the command line to mount multiple disks and tapes.
: 481      1141      2      ! However, it is necessary to reset DEVICE_INDEX for tape mounts because tape
: 482      1142      2      ! volumes are not mounted until every volume in the command line has been
: 483      1143      2      ! processed, and an error condition on the Nth volume will force all the work
: 484      1144      2      ! done on previous volumes to be undone.
: 485      1145      2
: 486      1146      2

```

```

487 1147 2 IF (.DEVICE_INDEX GTR 0) AND .STORED_CONTEXT[TAPE_MOUNT]
488 1148 THEN
489 1149     DEVICE_INDEX = 0;
490 1150
491 1151 2 INCR I FROM .DEVICE_INDEX TO .DEVICE_COUNT-1
492 1152 DO
493 1153     BEGIN
494 1154         : Mount the volume.  If the attempt failed, abort the mount
495 1155         : and return the error status.  Always dequeue the mount interlock(s),
496 1156         : no matter if the mount attempt succeeded or failed.
497 1157         :
498 1158     STATUS = VMOUNT ENVELOPE ();
499 1159     KERNEL CALL ( DEQ_MOUNT_LOCK );
500 1160     IF .LABLCK_STATUS[1] NEQ 0                               ! Dequeue the label lock if it exists
501 1161     THEN
502 1162         $DEQ ( LKID = .LABLCK_STATUS [1] );
503 1163
504 1164     IF NOT .STATUS
505 1165     THEN
506 1166         BEGIN
507 1167             $SETPRV (ENBFLG=0, PRVADR=MOUNT_PRIVS); ! Clear granted privileges
508 1168             $SETPRV (ENBFLG=1, PRVADR=USER_PRIVS); ! Restore old privileges
509 1169             RETURN (.STATUS);
510 1170         END;
511 1171     DEVICE_INDEX = .DEVICE_INDEX+1;
512 1172     END;
513 1173
514 1174     :
515 1175     : Deallocate all devices that are not mounted.
516 1176     :
517 1177     $DALLOC_DEVSSU (0);
518 1178
519 1179     :
520 1180     : Rebuild volume if mounting files-11 ODS-2 disk
521 1181     :
522 1182     :
523 1183     :
524 1184     IF .CLEANUP_FLAGS[CLF_REBUILD]
525 1185     THEN
526 1186         BEGIN
527 1187             STATUS = REBUILD_ENVELOPE ();
528 1188             $DASSGN (CHAN = .CHANNEL);                       ! Deassign channel used by REBUILD
529 1189             END;
530 1190
531 1191     :
532 1192     : If the rebuild was successful, attempt to activate the RUJ.
533 1193     :
534 1194     IF .STATUS
535 1195     THEN
536 1196         STATUS = ACTIVATE_JOURNAL ();
537 1197
538 1198     :
539 1199     : If the mount was successful, sent this mount request cluster-wide
540 1200     : when appropriate.
541 1201
542 1202     IF .STATUS
543 1203     THEN

```

```
544 1204 2 STATUS = MOUNT_CLUSTER (.ITEM_LIST); . Mount cluster-wide
545 1205 2
546 1206 2 ! Revoke any privileges that were granted.
547 1207 2
548 1208 2
549 1209 2 $$SETPRV (ENBFLG=0, PRVADR=MOUNT_PRIVS); ! Clear granted privileges
550 1210 2 $$SETPRV (ENBFLG=1, PRVADR=USER_PRIVS); ! Restore old privileges
551 1211 2
552 1212 2 RETURN (.STATUS)
553 1213 2
554 1214 2 ! end of routine MOUNT_COMMAND
END;
```

```
.TITLE VMOUNT
.IDENT \V04-002\
.PSECT $GLOBALS,NOEXE,2
```

```
00000 VMOUNT_GBL_START::
      .BLKB 0
00000 STORED_CONTEXT::
      .BLKB 4
00004 DATA_BASE_READY::
      .BLKB 4
00008 DEV_ALLOCATED::
      .BLKB 2
0000A .BLKB 2
0000C DEV_ACQUIRED::
      .BLKB 2
0000E .BLKB 2
00010 LOCK_STATUS::
      .BLKB 8
00018 CLEANUP_ALLOC::
      .BLKB 2
0001A .BLKB 2
0001C CLEANUP_FLAGS::
      .BLKB 4
00020 CHANNEL::
      .BLKB 4
00024 DEVICE_INDEX::
      .BLKB 4
00028 MAILBOX_CHANNEL::
      .BLKB 4
0002C CALLERS_ACMOD::
      .BLKB 4
00030 PHYS_COUNT::
      .BLKB 4
00034 PHYS_NAME::
      .BLKB 128
000B4 NAME_BUFFER::
      .BLKB 512
002B4 LOG_BUFFER::
      .BLKB 63
002F3 .BLKB 1
002F4 HOME_BLOCK::
      .BLKB 512
004F4 DEVICE_CHAR::
```

| | | | | | | | | |
|----|----|----|----|-------|------------------|--------|-----|-------|
| | | | | 00568 | DEVICE_CHAR2:: | .BLKB | 116 | |
| | | | | 005DC | HOMEBLOCK_LBN:: | .BLKB | 116 | |
| | | | | 005E0 | HEADER_LBN:: | .BLKB | 4 | |
| | | | | 005E4 | CURRENT_RVN:: | .BLKB | 4 | |
| | | | | 005E8 | USER_STATUS:: | .BLKB | 4 | |
| | | | | 005F0 | CURRENT_VCB:: | .BLKB | 8 | |
| | | | | 005F4 | REAL_MVL:: | .BLKB | 4 | |
| | | | | 005F8 | REAL_RVT:: | .BLKB | 4 | |
| | | | | 005FC | REAL_VCB:: | .BLKB | 4 | |
| | | | | 00600 | REAL_VCA:: | .BLKB | 4 | |
| | | | | 00604 | REAL_FCB:: | .BLKB | 4 | |
| | | | | 00608 | REAL_WCB:: | .BLKB | 4 | |
| | | | | 0060C | REAL_AQB:: | .BLKB | 4 | |
| | | | | 00610 | MTL_ENTRY:: | .BLKB | 4 | |
| | | | | 00614 | SMTL_ENTRY:: | .BLKB | 4 | |
| | | | | 00618 | MOUNT_ITMLST:: | .BLKB | 4 | |
| | | | | 0061C | LABLCK_STATUS:: | .BLKB | 4 | |
| | | | | 00624 | VMOUNT_GBL_END:: | .BLKB | 8 | |
| | | | | 00624 | ALLDEVNAM_BUF:: | .BLKB | 0 | |
| 24 | 55 | 4F | 4D | 00624 | ALLDEVNAM_BUF:: | .ASCII | | MOUS\ |
| | | | 20 | 00628 | | .ASCII | | \/ |
| | | | 20 | 00629 | | .ASCII | | \/ |
| | | | 20 | 0062A | | .ASCII | | \/ |
| | | | 20 | 0062B | | .ASCII | | \/ |
| | | | 20 | 0062C | | .ASCII | | \/ |
| | | | 20 | 0062D | | .ASCII | | \/ |
| | | | 20 | 0062E | | .ASCII | | \/ |
| | | | 20 | 0062F | | .ASCII | | \/ |
| | | | 20 | 00630 | | .ASCII | | \/ |
| | | | 20 | 00631 | | .ASCII | | \/ |
| | | | 20 | 00632 | | .ASCII | | \/ |
| | | | 20 | 00633 | | .ASCII | | \/ |
| | | | 20 | 00634 | | .ASCII | | \/ |
| | | | 20 | 00635 | | .ASCII | | \/ |
| | | | 20 | 00636 | | .ASCII | | \/ |
| | | | 20 | 00637 | | .ASCII | | \/ |
| | | | 20 | 00638 | | .ASCII | | \/ |
| | | | 20 | 00639 | | .ASCII | | \/ |

```
20 0063A .ASCII \\  
20 0063B .ASCII \\  
20 0063C .ASCII \\  
20 0063D .ASCII \\  
20 0063E .ASCII \\  
20 0063F .ASCII \\  
20 00640 .ASCII \\  
20 00641 .ASCII \\  
20 00642 .ASCII \\  
20 00643 .ASCII \\  
00000000 00644 ALLDEVNAM_DESC::  
          .LONG 0  
00000000' 00648 .ADDRESS ALLDEVNAM_BUF  
00000074 0064C DEVCHAR_DESC::  
          .LONG 116  
00000000' 00650 .ADDRESS DEVICE_CHAR  
00000074 00654 DEVCHAR_DESC2::  
          .LONG 116  
24 55 4F 4D 00000000' 00658 .ADDRESS DEVICE_CHAR2  
          0065C LABLCKNAM_BUF::  
          .ASCII \MOUS\  
20 00660 .ASCII \\  
20 00661 .ASCII \\  
20 00662 .ASCII \\  
20 00663 .ASCII \\  
20 00664 .ASCII \\  
20 00665 .ASCII \\  
20 00666 .ASCII \\  
20 00667 .ASCII \\  
20 00668 .ASCII \\  
20 00669 .ASCII \\  
20 0066A .ASCII \\  
20 0066B .ASCII \\  
20 0066C .ASCII \\  
20 0066D .ASCII \\  
20 0066E .ASCII \\  
20 0066F .ASCII \\  
20 00670 .ASCII \\  
20 00671 .ASCII \\  
20 00672 .ASCII \\  
20 00673 .ASCII \\  
20 00674 .ASCII \\  
20 00675 .ASCII \\  
20 00676 .ASCII \\  
20 00677 .ASCII \\  
20 00678 .ASCII \\  
20 00679 .ASCII \\  
20 0067A .ASCII \\  
20 0067B .ASCII \\  
20 0067C .ASCII \\  
20 0067D .ASCII \\  
20 0067E .ASCII \\  
20 0067F .ASCII \\  
00000000 00680 LABLCKNAM_DESC::  
          .LONG 0  
00000000' 00684 .ADDRESS LABLCKNAM_BUF
```


VOL1==

HOME_BLOCK

```

.EXTRN ACTIVATE JOURNAL
.EXTRN $DALLOC DEV$$, CHECK PARAMS
.EXTRN DEVICE COUNT, LCK GLOBAL START
.EXTRN LCK GLOBAL END, SYSS$SETPRV
.EXTRN SYSS$CMKRNL, SYSS$DEQ
.EXTRN SYSS$DASSGN

```

.PSECT \$CODE\$,NOWRT,2

```

.ENTRY SYSS$VMOUNT, Save R2,R3,R4,R5,R6,R7 : 1033
MOVAB SYSS$SETPRV, R7
MOVAB DEVICE_INDEX, R6
SUBL2 #16, SP
CLRL CHANNEL : 1093
MOVL #1, USER STATUS : 1094
MOVL #1623978784, MOUNT_PRIVS : 1096
BISB2 #32, MOUNT_PRIVS+4 : 1111
BLBS DATA_BASE_READY, 2$ : 1120
MOVCS #0, (SP), #0, #1572, VMOUNT_GBL_START : 1123

MOVCS #0, (SP), #0, #LCK GLOBAL END-- : 1124
LCK GLOBAL_START, [LCK GLOBAL_START
MOVL ITEM_LIST, MOUNT_ITMLST : 1125
MOVL #1, DATA_BASE_READY : 1126
CALLG (AP), CHECK_PARAMS : 1127
MOVL R0, STATUS
BLBS R0, 1$
BRW 9$
MOVPSL CURRENT_PSL : 1130
EXTZV #22, #2, CURRENT_PSL, CALLERS_ACMOD : 1131
PUSHL SP : 1138
CLRL -(SP)
PUSHAB MOUNT_PRIVS
PUSHL #1
CALLS #4, SYSS$SETPRV
TSTL DEVICE_INDEX : 1147
BLEQ 3$
BLBC STORED_CONTEXT, 3$
CLRL DEVICE_INDEX : 1149
MOVL DEVICE_COUNT, R4 : 1151
SUBL3 #1, DEVICE_INDEX, 1
BRB 6$
CALLS #0, VMOUNT_ENVELOPE : 1159
MOVL R0, STATUS
CLRL -(SP) : 1160
PUSHL SP
PUSHAB DEQ_MOUNT_LOCK
CALLS #3, @SYSS$CMKRNL
MOVL LABLCK_STATUS+4, R0 : 1161
BEQL 5$
CLRQ -(SP) : 1163
CLRL -(SP)
PUSHL R0
CALLS #4, SYSS$DEQ : 1165
BLBC STATUS, 8$ : 1165
INCL DEVICE_INDEX : 1172

```

```

00FC 00000
57 00000000G 00 9E 00002
56 00000000' EF 9E 00009
5E 10 C2 00010
FC A6 D4 00013
05C4 C6 01 D0 00016
08 AE 60CBF320 8F D0 0001B
OC AE 20 88 00023
35 E0 A6 EB 00027
6E 00 2C 0002B
DC A6 00 00032
6E 00 2C 00034
00000000G 00 00038
05F4 C6 04 AC D0 00040
EO A6 01 D0 00046
C000G CF 6C FA 0004A
52 50 D0 0004F
03 50 EB 00052
00B6 31 00055
50 DC 00058 1$:
08 A6 50 02 16 EF 0005A
5E DD 00060 2$:
7E D4 00062
10 AE 9F 00064
01 DD 00067
67 04 FB 00069
66 D5 0006C
06 15 0006E
02 DC A6 E9 00070
66 D4 00074
54 00000000G 00 D0 00076 3$:
66 01 C3 0007D
34 11 00081
00000000V EF 00 FB 00083 4$:
52 50 D0 0008A
7E D4 0008D
5E DD 0008F
00000000V EF 9F 00091
00000000G 9F 03 FB 00097
50 05FC C6 D0 0009E
0D 13 000A3
7E 7C 000A5
7E D4 000A7
50 DD 000A9
00000000G 00 04 FB 000AB
45 52 E9 000B2 5$:
66 D6 000B5

```

| | | | | | | | | | |
|----|-----------|----|----|-------------|------|--------|--------------------------|---|------|
| C8 | 53 | 54 | F2 | 000B7 | 6\$: | A0BLSS | R4, I, 4\$ | : | 1151 |
| | | 7E | D4 | 000BB | | CLRL | -(SP) | : | 1178 |
| 14 | 00000000G | 00 | 01 | FB 000BD | | CALLS | #1, \$DALLOC_DEVSSU | : | |
| | F9 | A6 | 01 | E1 000C4 | | BBC | #1, CLEANUP_FLAGS+1, 7\$ | : | 1184 |
| | 00000000V | EF | 00 | FB 000C9 | | CALLS | #0, REBUILD_ENVELOPE | : | 1187 |
| | | 52 | 50 | DD 000D0 | | MOVL | R0, STATUS | : | |
| | | | FC | A6 DD 000D3 | | PUSHL | CHANNEL | : | 1188 |
| | 00000000G | 00 | 01 | FB 000D6 | | CALLS | #1, SYSSDASSGN | : | |
| | | 1A | 52 | E9 000DD | 7\$: | BLBC | STATUS, 8\$ | : | 1194 |
| | 00000000G | 00 | 00 | FB 000E0 | | CALLS | #0, ACTIVATE_JOURNAL | : | 1196 |
| | | 52 | 50 | DD 000E7 | | MOVL | R0, STATUS | : | |
| | | 0D | 52 | E9 000EA | | BLBC | STATUS, 8\$ | : | 1202 |
| | | | 04 | AC DD 000ED | | PUSHL | ITEM LIST | : | 1204 |
| | 00000000V | EF | 01 | FB 000F0 | | CALLS | #1, MOUNT_CLUSTER | : | |
| | | 52 | 50 | DD 000F7 | | MOVL | R0, STATUS | : | |
| | | | 7E | 7C 000FA | 8\$: | CLRQ | -(SP) | : | 1209 |
| | | | 10 | AE 9F 000FC | | PUSHAB | MOUNT_PRIVS | : | |
| | | 67 | 7E | D4 000FF | | CLRL | -(SP) | : | |
| | | | 04 | FB 00101 | | CALLS | #4, SYSSSETPRV | : | |
| | | | 7E | 7C 00104 | | CLRQ | -(SP) | : | 1210 |
| | | | 08 | AE 9F 00106 | | PUSHAB | USER_PRIVS | : | |
| | | 67 | 01 | DD 00109 | | PUSHL | #1 | : | |
| | | 50 | 04 | FB 0010B | | CALLS | #4, SYSSSETPRV | : | |
| | | | 52 | DD 0010E | 9\$: | MOVL | STATUS, R0 | : | 1212 |
| | | | 04 | 00111 | | RET | | : | 1214 |

; Routine Size: 274 bytes, Routine Base: \$CODE\$ + 0000

```

556 1215 1 ROUTINE VMOUNT_ENVELOPE =
557 1216 1
558 1217 1 ++
559 1218 1
560 1219 1 FUNCTIONAL DESCRIPTION:
561 1220 1
562 1221 1 This routine serves as the base call frame for all the EXEC
563 1222 1 mode code, and provides a convenient (and necessary) spot
564 1223 1 from which to intercept all EXEC mode conditions.
565 1224 1
566 1225 1 CALLING SEQUENCE:
567 1226 1
568 1227 1 This routine should be called in EXEC mode.
569 1228 1
570 1229 1 INPUT:
571 1230 1
572 1231 1 None.
573 1232 1
574 1233 1 OUTPUT:
575 1234 1
576 1235 1 None.
577 1236 1
578 1237 1 IMPLICIT INPUTS:
579 1238 1
580 1239 1 Current mode is EXEC, DEVICE_INDEX contains an integer value.
581 1240 1
582 1241 1 ROUTINE VALUE:
583 1242 1
584 1243 1 This routine returns the status returned by MOUNT_VOLUME.
585 1244 1
586 1245 1 --
587 1246 1
588 1247 2 BEGIN
589 1248 2
590 1249 2 LOCAL
591 1250 2 STATUS;
592 1251 2
593 1252 2 |
594 1253 2 | Establish the special EXEC mode condition handler.
595 1254 2
596 1255 2 ENABLE INTERCEPT_SIGNAL;
597 1256 2
598 1257 2 |
599 1258 2 | Attempt to mount the volume.
600 1259 2
601 1260 2 STATUS = MOUNT_VOLUME (.DEVICE_INDEX);
602 1261 2
603 1262 2 RETURN (.STATUS)
604 1263 3
605 1264 1 END;

```

0000 0000 VMOUNT_ENVELOPE:
.WORD Save nothing

; 1215

| | | | | | | | | | | |
|-----------|----|-----------|----|------|-------|------|-------|----------------------|---|------|
| | 6D | 000F | CF | DE | 00002 | | MOVAL | 1\$, (FP) | : | 1247 |
| 00000000V | EF | 00000000' | EF | DD | 00007 | | PUSHL | DEVICE_INDEX | : | 1260 |
| | | | 01 | FB | 0000D | | CALLS | #1, MOUNT_VOLUME | : | |
| | | | | 04 | 00014 | | RET | | : | 1264 |
| | | | | 0000 | 00015 | 1\$: | .WORD | Save nothing | : | 1247 |
| | | | 7E | D4 | 00017 | | CLRL | -(SP) | : | |
| | 7E | 04 | 5E | DD | 00019 | | PUSHL | SP | : | |
| 00000000V | EF | | AC | 7D | 0001B | | MOVQ | 4(AP), -(SP) | : | |
| | | | 03 | FB | 0001F | | CALLS | #3, INTERCEPT_SIGNAL | : | |
| | | | 04 | 04 | 00026 | | RET | | : | |

; Routine Size: 39 bytes, Routine Base: \$CODE\$ + 0112

```

607 1265 1 ROUTINE REBUILD_ENVELOPE =
608 1266 1
609 1267 1 :++
610 1268 1
611 1269 1 FUNCTIONAL DESCRIPTION:
612 1270 1
613 1271 1 This routine serves as the base call frame for all the EXEC
614 1272 1 mode code, and provides a convenient (and necessary) spot
615 1273 1 from which to intercept all EXEC mode conditions.
616 1274 1
617 1275 1 CALLING SEQUENCE:
618 1276 1
619 1277 1 This routine should be called in EXEC mode.
620 1278 1
621 1279 1 INPUT:
622 1280 1
623 1281 1 None.
624 1282 1
625 1283 1 OUTPUT:
626 1284 1
627 1285 1 None.
628 1286 1
629 1287 1 IMPLICIT INPUTS:
630 1288 1
631 1289 1 Current mode is EXEC, DEVICE_INDEX contains an integer value.
632 1290 1
633 1291 1 ROUTINE VALUE:
634 1292 1
635 1293 1 This routine returns the status returned by MOUNT_VOLUME.
636 1294 1
637 1295 1 --
638 1296 1
639 1297 2 BEGIN
640 1298 2
641 1299 2 EXTERNAL ROUTINE
642 1300 2 STAND_ALONE_REBUILD; ! Rebuild quota file and bitmaps (ODS2)
643 1301 2
644 1302 2 LOCAL
645 1303 2 STATUS;
646 1304 2
647 1305 2 :
648 1306 2 Establish the special EXEC mode condition handler.
649 1307 2
650 1308 2 ENABLE INTERCEPT_SIGNAL;
651 1309 2
652 1310 2 :
653 1311 2 Rebuild the volume.
654 1312 2 :
655 1313 2 ERR MESSAGE (MOUN$ REBUILD);
656 P 1314 2 STATUS = $ASSIGN (DEVNAM = PHYS NAME[0],
657 1315 2 CHAN = CHANNEL);
658 1316 2 IF NOT .STATUS THEN ERR EXIT (.STATUS);
659 1317 2 STAND_ALONE_REBUILD (.CHANNEL);
660 1318 2
661 1319 2 RETURN 1
662 1320 2
663 1321 1 END;

```

.EXTRN STAND_ALONE_REBUILD
.EXTRN SYSS\$ASSIGN

0004 00000 REBUILD_ENVELOPE:

| | | | | | | | | | |
|-----------|----|-----------|----|-------|-------|--------|-------------------------|---|------|
| | 52 | 00000000' | EF | 9E | 00002 | .WORD | Save R2 | : | 1265 |
| | 6D | 0033 | CF | DE | 00009 | MOVAB | CHANNEL, R2 | : | 1297 |
| | | 0072A01B | 8F | DD | 0000E | MOVAL | 2\$(FP) | : | 1313 |
| 00000000G | 00 | | 01 | FB | 00014 | PUSHL | #7\$12091 | : | |
| | | | 7E | 7C | 0001B | CALLS | #1, LIB\$SIGNAL | : | |
| | | | 52 | DD | 0001D | CLRQ | -(SP) | : | 1315 |
| | | 14 | A2 | 9F | 0001F | PUSHL | R2 | : | |
| 00000000G | 00 | | 04 | FB | 00022 | PUSHAB | PHYS_NAME | : | |
| | 09 | | 50 | E8 | 00029 | CALLS | #4, SYSS\$ASSIGN | : | |
| | | | 50 | DD | 0002C | BLBS | STATUS, 1\$ | : | 1316 |
| 00000000G | 00 | | 01 | FB | 0002E | PUSHL | STATUS | : | |
| | | | 62 | DD | 00035 | CALLS | #1, LIB\$STOP | : | |
| 0000G | CF | | 01 | FB | 00037 | PUSHL | CHANNEL | : | 1317 |
| | 50 | | 01 | D0 | 0003C | CALLS | #1, STAND_ALONE_REBUILD | : | |
| | | | 04 | 0003F | | MOVL | #1, R0 | : | 1319 |
| | | | | 0000 | 00040 | RET | | : | 1321 |
| | | | 7E | D4 | 00042 | .WORD | Save nothing | : | 1297 |
| | | | 5E | DD | 00044 | CLRL | -(SP) | : | |
| | 7E | 04 | AC | 7D | 00046 | PUSHL | SP | : | |
| 00000000V | EF | | 03 | FB | 0004A | MOVQ | 4(AP), -(SP) | : | |
| | | | 04 | 00051 | | CALLS | #3, INTERCEPT_SIGNAL | : | |
| | | | | | | RET | | : | |

; Routine Size: 82 bytes, Routine Base: \$CODE\$ + 0139

```

665 1322 1 ROUTINE INTERCEPT_SIGNAL (SIGNAL, MECHANISM) =
666 1323 1
667 1324 1 +-
668 1325 1 Functional Description:
669 1326 1
670 1327 1 This routine is a conditon handler whose sole
671 1328 1 reason for existence is to force the primary
672 1329 1 conditon code's facility-code to that of the
673 1330 1 MOUNT facility.
674 1331 1
675 1332 1 Input:
676 1333 1
677 1334 1 SIGNAL = Address of the signal array
678 1335 1 MECHANISM = Address of the mechanism array
679 1336 1
680 1337 1 Output:
681 1338 1
682 1339 1 The condition facility code is equal to MOUN$_FACILITY
683 1340 1 --
684 1341 1
685 1342 2 BEGIN ! Start of INTERCEPT_SIGNAL
686 1343 2
687 1344 2 MAP
688 1345 2
689 1346 2 SIGNAL : REF BBLOCK, ! Signal array
690 1347 2 MECHANISM : REF BBLOCK; ! Mechanism array
691 1348 2
692 1349 2 EXTERNAL
693 1350 2
694 1351 2 MOUNT_OPTIONS : ADDRESSING_MODE (GENERAL)
695 1352 2 BITVECTOR VOLATILE, ! parser option flags
696 1353 2 USER_STATUS : VECTOR, ! Status return of some routines
697 1354 2 VOLINV_COUNT : ADDRESSING_MODE (GENERAL);
698 1355 2 ! VOLINV retry counter
699 1356 2
700 1357 2 EXTERNAL LITERAL
701 1358 2 VOLINV_LIMIT; ! VOLINV retry limit
702 1359 2
703 1360 2
704 1361 2 IF .SIGNAL[CHFS$_SIG_NAME] NEQ S$_UNWIND
705 1362 2 THEN
706 1363 2 BEGIN
707 1364 2 |
708 1365 2 | Make the facility code MOUN$_FCILITY.
709 1366 2 |
710 1367 2 IF .BBLOCK [SIGNAL[CHFS$_SIG_NAME], STSSV_FAC_NO] EQL 0
711 1368 2 OR .BBLOCK [SIGNAL[CHFS$_SIG_NAME], STSSV_FAC_NO] EQL INITS_FACILITY
712 1369 2 THEN
713 1370 2 BBLOCK [SIGNAL[CHFS$_SIG_NAME], STSSV_FAC_NO] = MOUN$_FACILITY;
714 1371 2
715 1372 2 IF .BBLOCK [SIGNAL[CHFS$_SIG_NAME], STSSV_MSG_NO] EQL 0
716 1373 2 THEN
717 1374 2 BBLOCK [SIGNAL[CHFS$_SIG_NAME], STSSV_MSG_NO] = .USER_STATUS [0] ^ (-$BITPOSITION (STSSV_MSG_NO));
718 1375 2
719 1376 2 |
720 1377 2 | If the caller requested it, print the message text associated with the
721 1378 2 | message. Also make sure that the particular error is not covered by

```

```

: 722 1379 3 ! operator assisted mount. If it is, do not print the message.
: 723 1380 3
: 724 1381 5 IF (.MOUNT_OPTIONS [OPT_MESSAGE] AND NOT (.MOUNT_OPTIONS [OPT_ASSIST]
: 725 1382 6 AND (SELECTONEU (.SIGNAL [CHF$L_SIG_NAME] AND STS$M_MSG_NO) OF
: 726 1383 6 SET
: 727 1384 6 [SS$_DEVALLOC AND STS$M_MSG_NO] : 1;
: 728 1385 6 [SS$_MEDOFL AND STS$M_MSG_NO] : 1;
: 729 1386 6 [SS$_VOLINV AND STS$M_MSG_NO] : 1;
: 730 1387 6 [SS$_NODEVAVL AND STS$M_MSG_NO] : 1;
: 731 1388 6 [SS$_NOSUCHDEV AND STS$M_MSG_NO] : 1;
: 732 1389 6 [SS$_INCVOLLABEL AND STS$M_MSG_NO] : 1;
: 733 1390 6 [OTHERWISE] : 0;
: 734 1391 4 TES)))
: 735 1392 4
: 736 1393 4 !
: 737 1394 4 ! If mounting with /NOASSIST and we are in VOLINV retry, suppress outputting
: 738 1395 4 ! the VOLINV error message unless this is the last retry attempt.
: 739 1396 4 !
: 740 1397 5 AND (.MOUNT_OPTIONS [OPT_MESSAGE] AND NOT (NOT .MOUNT_OPTIONS [OPT_ASSIST]
: 741 1398 6 AND (SELECTONEU (.SIGNAL [CHF$L_SIG_NAME] AND STS$M_MSG_NO) OF
: 742 1399 6 SET
: 743 1400 6 [SS$_VOLINV AND STS$M_MSG_NO] : IF .VOLINV_COUNT LSS VOLINV_LIMIT-1
: 744 1401 6 THEN
: 745 1402 6 1
: 746 1403 6 ELSE
: 747 1404 6 0;
: 748 1405 6
: 749 1406 6 [OTHERWISE] : 0;
: 750 1407 4 TES)))
: 751 1408 3 THEN
: 752 1409 4 BEGIN
: 753 1410 4 SIGNAL [CHF$L_SIG_ARGS] = .SIGNAL [CHF$L_SIG_ARGS] - 2;
: 754 1411 4 $PUTMSG (MSGVEC = SIGNAL [CHF$L_SIG_ARGS], ACTRTN=0, FACNAM=0);
: 755 1412 4 SIGNAL [CHF$L_SIG_ARGS] = .SIGNAL [CHF$L_SIG_ARGS] + 2;
: 756 1413 4 BBLOCK [SIGNAL [CHF$L_SIG_NAME], STS$V_INHIB_MSG] = 1;
: 757 1414 3 END;
: 758 1415 3 !
: 759 1416 3 !
: 760 1417 3 ! If the condition severity code is SEVERE or ERROR, then unwind the
: 761 1418 3 ! stack back to the caller of the frame that established this handler.
: 762 1419 3 ! Return the condition code in R0.
: 763 1420 3 !
: 764 1421 3 IF .BBLOCK [SIGNAL [CHF$L_SIG_NAME], STS$V_SEVERITY] EQL STS$K_SEVERE
: 765 1422 3 OR .BBLOCK [SIGNAL [CHF$L_SIG_NAME], STS$V_SEVERITY] EQL STS$K_ERROR
: 766 1423 3 THEN
: 767 1424 4 BEGIN
: 768 1425 4 MECHANISM [CHF$L_MCH_SAVRO] = .SIGNAL [CHF$L_SIG_NAME];
: 769 1426 4 $UNWIND ();
: 770 1427 3 END;
: 771 1428 2 END;
: 772 1429 2 !
: 773 1430 2 !
: 774 1431 2 ! Attempt to continue the operation.
: 775 1432 2 !
: 776 1433 2 RETURN (SS$_CONTINUE);
: 777 1434 2 !
: 778 1435 1 END; ! End of INTERCEPT_SIGNAL

```


.EXTRN MOUNT_OPTIONS, VOLINV_COUNT
.EXTRN VOLINV_LIMIT, SYSS\$PUTMSG
.EXTRN SYSS\$UNWIND

001C 00000 INTERCEPT SIGNAL:

| | | | | | | | | | | | |
|----------|----|----|----|-----------|-------|------|----------|----------|-------|-------------------------------|------|
| | | | 54 | 00000000G | 00 | 9E | 00002 | | .WORD | Save R2,R3,R4 | 1322 |
| | | | 52 | | 04 | AC | D0 | 00009 | MOVAB | MOUNT_OPTIONS+6, R4 | |
| | | | 53 | | 04 | A2 | 9E | 0000D | MOVL | SIGNAL, R2 | 1361 |
| | | | 8F | 00000920 | | 63 | D1 | 00011 | MOVAB | 4(R2), R3 | |
| | | | | | | 03 | 12 | 00018 | CMPL | (R3), #2336 | |
| | | | | | | 00D6 | 31 | 0001A | BNEQ | 1\$ | |
| | | | | OFFF | 8F | 02 | A3 | B3 | BRW | 9\$ | |
| | | | | | | 0C | 13 | 00023 | BITW | 2(R3), #4095 | 1367 |
| 00000075 | 8F | 02 | A3 | | 0C | 00 | ED | 00025 | BEQL | 2\$ | |
| | | | | | | 0A | 12 | 0002F | CMPZV | #0, #12, 2(R3), #117 | 1368 |
| | 02 | A3 | 0C | | 00 | 0A | 12 | 0002F | BNEQ | 3\$ | |
| | | | | FFF8 | 8F | 00 | 00000072 | 8F | INSV | #114, #0, #12, 2(R3) | 1370 |
| | | | | | | 63 | B3 | 0003B | BITW | (R3), #65528 | 1372 |
| | | | | | | 0C | 12 | 00040 | BNEQ | 4\$ | |
| | | | | 50 | 0000G | CF | FD | 8F | ASHL | #-3, USER_STATUS, R0 | 1374 |
| | | | | | | 03 | 50 | F0 | INSV | R0, #3, #T3, (R3) | |
| | 63 | | | | | 01 | 03 | EF | EXTZV | #3, #1, MOUNT_OPTIONS+6, R1 | 1381 |
| | 51 | | | | | 7E | 51 | E9 | BLBC | R1, 7\$ | |
| | | | | | | 64 | 01 | E1 | BBC | #2, MOUNT_OPTIONS+6, 5\$ | |
| | | | | | | 3E | 01 | E1 | BICL3 | #-65529, (R3), R0 | 1382 |
| | | | | | | 50 | 63 | FFFF0007 | CMPL | R0, #2112 | 1384 |
| | | | | 00000840 | 8F | 51 | D1 | 00062 | BEQL | 7\$ | |
| | | | | | | 69 | 13 | 00069 | CMPL | R0, #416 | 1385 |
| | | | | 000001A0 | 8F | 50 | D1 | 0006B | BEQL | 7\$ | |
| | | | | | | 60 | 13 | 00072 | CMPL | R0, #592 | 1386 |
| | | | | 00000250 | 8F | 50 | D1 | 00074 | BEQL | 7\$ | |
| | | | | | | 57 | 13 | 0007B | CMPL | R0, #2480 | 1387 |
| | | | | 00000980 | 8F | 50 | D1 | 0007D | BEQL | 7\$ | |
| | | | | | | 4E | 13 | 00084 | CMPL | R0, #2312 | 1388 |
| | | | | 00000908 | 8F | 50 | D1 | 00086 | BEQL | 7\$ | |
| | | | | | | 45 | 13 | 0008D | CMPL | R0, #264 | 1389 |
| | | | | 00000108 | 8F | 50 | D1 | 0008F | BEQL | 7\$ | |
| | | | | | | 3C | 13 | 00096 | CMPL | R0, #264 | |
| | | | | | | 39 | 51 | E9 | BLBC | R1, 7\$ | 1397 |
| | | | | | | 64 | 02 | E0 | BBS | #2, MOUNT_OPTIONS+6, 6\$ | |
| | | | | | | 50 | 63 | FFFF0007 | BICL3 | #-65529, (R3), R0 | 1398 |
| | | | | 00000250 | 8F | 50 | D1 | 000A7 | CMPL | R0, #592 | 1400 |
| | | | | | | 0D | 12 | 000AE | BNEQ | 6\$ | |
| | | | | 00000000G | 8F | 00 | D1 | 000B0 | CMPL | VOLINV_COUNT, #VOLINV_LIMIT-1 | |
| | | | | | | 17 | 19 | 000BB | BLSS | 7\$ | |
| | | | | | | 62 | 02 | C2 | SUBL2 | #2, (R2) | 1410 |
| | | | | | | 7E | 7C | 000C0 | CLRQ | -(SP) | 1411 |
| | | | | | | 7E | D4 | 000C2 | CLRL | -(SP) | |
| | | | | | | 52 | DD | 000C4 | PUSHL | R2 | |
| | | | | 00000000G | 00 | 04 | FB | 000C6 | CALLS | #4, SYSS\$PUTMSG | |
| | | | | | | 62 | 02 | C0 | ADDL2 | #2, (R2) | 1412 |
| | | | | | | 03 | A3 | 000CD | BISB2 | #16, 3(R3) | 1413 |
| | 04 | | | | | 03 | 00 | ED | CMPZV | #0, #3, (R3), #4 | 1421 |
| | | | | | | 07 | 13 | 000D9 | BEQL | 8\$ | |
| | 02 | | | | | 03 | 00 | ED | CMPZV | #0, #3, (R3), #2 | 1422 |

VMOUNT
V04-002

| | | | | | | | | | | | |
|-----------|----|----|----|-------|-------|------|-------|-----------------|--|---|------|
| | | | 11 | 12 | 000E0 | | BNEQ | 9\$ | | : | |
| | 0C | 50 | AC | D0 | 000E2 | 8\$: | MOVL | MECHANISM, R0 | | : | 1425 |
| | | A0 | 63 | D0 | 000E6 | | MOVL | (R3), 12(R0) | | : | |
| | | | 7E | 7C | 000EA | | CLRQ | -(SP) | | : | 1426 |
| 00000000G | | 00 | 02 | FB | 000EC | | CALLS | #2, SYS\$UNWIND | | : | |
| | | 50 | 01 | D0 | 000F3 | 9\$: | MOVL | #1, R0 | | : | 1433 |
| | | | 04 | 000F6 | | | RET | | | : | 1435 |

; Routine Size: 247 bytes, Routine Base: \$CODE\$ + 018B

V
V

```

: 780      1436 1 ROUTINE MC'INT_VOLUME (J) =
: 781      1437 1
: 782      1438 1 |++
: 783      1439 1 |
: 784      1440 1 | FUNCTIONAL DESCRIPTION:
: 785      1441 1 |
: 786      1442 1 |     This routine will mount a single disk or tape volume.
: 787      1443 1 |
: 788      1444 1 | CALLING SEQUENCE:
: 789      1445 1 |
: 790      1446 1 |     mount_volume (.j)
: 791      1447 1 |
: 792      1448 1 | INPUT:
: 793      1449 1 |
: 794      1450 1 |     J : Index into device list.
: 795      1451 1 |
: 796      1452 1 | OUTPUT:
: 797      1453 1 |
: 798      1454 1 |     None.
: 799      1455 1 |
: 800      1456 1 | IMPLICIT INPUT:
: 801      1457 1 |
: 802      1458 1 |     Mount data base
: 803      1459 1 |
: 804      1460 1 | IMPLICIT OUTPUT:
: 805      1461 1 |
: 806      1462 1 |     None.
: 807      1463 1 |
: 808      1464 1 | ROUTINE VALUE:
: 809      1465 1 |
: 810      1466 1 |     Assorted status codes.
: 811      1467 1 |
: 812      1468 1 | SIDE EFFECTS:
: 813      1469 1 |
: 814      1470 1 |     Volume mounted, device data base updated.
: 815      1471 1 | --
: 816      1472 1 |
: 817      1473 2 BEGIN
: 818      1474 2
: 819      1475 2 LOCAL
: 820      1476 2     DEVICE_ITMLST1 : BBLOCK [(1 * 12) + 4] INITIAL
: 821      1477 2     |
: 822      1478 2     | item: allocation class plus device name
: 823      1479 2     |
: 824      1480 2     | (WORD (NAMEBUF LEN-4),
: 825      1481 2     | WORD (DVIS_ALCDEVNAM),
: 826      1482 2     | LONG (ALLDEVNAM_BUF+4),
: 827      1483 2     | LONG (ALLDEVNAM_DESC),
: 828      1484 2     |
: 829      1485 2     | end of list
: 830      1486 2     |
: 831      1487 2     | LONG (0)),
: 832      1488 2     |
: 833      1489 2     | P      : string scan pointer
: 834      1490 2     | STATUS: : system service status
: 835      1491 2
: 836      1492 2 EXTERNAL
:           DEV_CTX      : BBLOCK FIELD (DC), ! device value block context fields

```

```

837 1493 MOUNT_FAILED : ADDRESSING_MODE (GENERAL) LONG VOLATILE, ! State of the current mount
838 1494 MOUNT_OPTIONS : ADDRESSING_MODE (GENERAL) BITVECTOR VOLATILE, ! parser option flags
839 1495 DEVICE_COUNT : ADDRESSING_MODE (GENERAL), ! number of devices specified
840 1496 LABEL_COUNT : ADDRESSING_MODE (GENERAL), ! number of volume labels specified
841 1497 DEVICE_STRING : ADDRESSING_MODE (GENERAL) VECTOR VOLATILE, ! device name string descriptor
842 1498 LABEL_STRING : ADDRESSING_MODE (GENERAL) VECTOR VOLATILE; ! volume label string descriptor
843 1499
844 1500 EXTERNAL ROUTINE
845 1501
846 1502 SEARCH_VOL, ! search I/O database for volume
847 1503 TRAN_LOGNAME, ! translate logical name
848 1504 READ_VOLLABEL, ! read magtape volume header label
849 1505 READ_HOMEBLOCK, ! read disk home block
850 1506 MOUNT_TAPE ! mount magtape
851 1507 MOUNT_DISK1, ! mount level 1 disk
852 1508 MOUNT_DISK2, ! mount level 1 disk
853 1509 GET_DEVICE_CONTEXT; ! get device lock value block context
854 1510
855 1511 BIND
856 1512 OPTIONS = MOUNT_OPTIONS : VECTOR VOLATILE;
857 1513
858 1514
859 1515 ENABLE MAIN_HANDLER; ! Enable the MOUNT condition handler
860 1516
861 1517
862 1518 ! Reset the mount options bit mask.
863 1519
864 1520
865 1521 OPTIONS[0] = .OPTIONS[0] AND NOT RESET_OPTIONS1;
866 1522 OPTIONS[1] = .OPTIONS[1] AND NOT RESET_OPTIONS2;
867 1523 MOUNT_FAILED = 1;
868 1524
869 1525 BEGIN
870 1526
871 1527 ! rebind things to make life easier ( so we see them as their
872 1528 ! real logical units)
873 1529
874 1530 MAP
875 1531 DEVICE_STRING : BBLOCKVECTOR [ DEVMAX, 8 ],
876 1532 NAME_BUFFER : BBLOCKVECTOR [ DEVMAX, NAMEBUF_LEN ],
877 1533 PHYS_NAME : BBLOCKVECTOR [ DEVMAX, 8 ];
878 1534
879 1535
880 1536 ! Start of buffer
881 1537
882 1538 MACRO STADR = 0,0,0,0%;
883 1539
884 1540
885 1541 ! Define descriptor vector displacements
886 1542
887 1543 MACRO LEN = 0,0,32,0%;
888 1544 MACRO ADDR = 4,0,32,0%;
889 1545 MACRO ILEN = 8,0,32,0%; ! Item list returned length position.
890 1546
891 1547
892 1548
893 1549 ! If the device is being mounted /SHARE, /GROUP, or /SYSTEM, search the

```

```

894 1550 3 ! device database for a matching volume label. To properly serialize
895 1551 3 ! simultaneous shared mounts take out the label lock in E mode. This
896 1552 3 ! label lock will be released in routine SYSSVMOUNT when everything is
897 1553 3 ! done.
898 1554 3
899 1555 3 STATUS = 0;
900 1556 3 IF NOT .MOUNT_OPTIONS [OPT_NOSHARE]
901 1557 3 THEN
902 1558 3 BEGIN
903 1559 3
904 1560 3
905 1561 3 ! The label lock has the form MOUS-csid-vollabel. The csid part makes
906 1562 3 ! the label lock node-specific, which is necessary to avoid potential
907 1563 3 ! deadlocks in a cluster. If the node is not in a cluster, the csid
908 1564 3 ! field is set to zero.
909 1565 3
910 1566 3 LOCAL
911 1567 3     CSID          : LONG INITIAL (0),      ! Initialize to zero
912 1568 3     SYI_ITMLST   : BLOCK [(1*12)+4, BYTE] INITIAL
913 1569 3     ( WORD (4),   ! Return buffer length
914 1570 3     WORD (SYI$ NODE_CSID), ! CSID item code
915 1571 3     LONG (CSID),   ! Return buffer address
916 1572 3     LONG (0),
917 1573 3     LONG (0));
918 1574 3
919 P 1575 3 $GETSYIW ( EFN = MOUNT_EFN,          ! Get CSID of the local node
920 1576 3           ITMLST = SYI_ITMLST );
921 1577 3
922 1578 3
923 1579 3 ! Set up the label lock resource name and descriptor
924 1580 3
925 1581 3 LABLCKNAM_DESC [0] = .LABEL_STRING [.J*2] + 8; ! 'MOUS' prefix + CSID
926 1582 3 LABLCKNAM_BUF + 4 = .CSID;                   ! Merge in CSID
927 1583 3 CH$COPY 7 .LABEL_STRING [.J*2],             ! Length of input string
928 1584 3          .LABEL_STRING [.J*2+1],           ! Address of label string buffer
929 1585 3          0,
930 1586 3          .LABEL_STRING [.J*2],             ! Length of output string
931 1587 3          LABLCKNAM_BUF + 8 );               ! Address of output buffer
932 1588 3
933 P 1589 3 $ENQW ( LKMODE = LCK$K_EXMODE,          ! Take out the label lock
934 P 1590 3         LKSB = LABLCKR_STATUS,
935 P 1591 3         FLAGS = LCK$M_SYSTEM,
936 P 1592 3         RESNAM = LABLCKNAM_DESC,
937 P 1593 3         EFN = MOUNT_EFN,
938 1594 3         ACMODE = PSL$C_EXEC );
939 1595 3
940 1596 3 STATUS = KERNEL_CALL (SEARCH_VOL, LABEL_STRING[.J*2]);
941 1597 3 END;
942 1598 3
943 1599 3
944 1600 3 ! The SEARCH_VOL routine will only return success if this is a /SHARE
945 1601 3 ! mount and a matching volume label is found. It will signal an error
946 1602 3 ! if this is a /SYSTEM or /GROUP mount and a duplicate volume label is
947 1603 3 ! already in use.
948 1604 3
949 1605 3 IF .STATUS
950 1606 3 !

```

```

: 951      1607 3      | A successful /SHARE mount. Just print the message here; we rejoin
: 952      1608 3      | the "volume not found" path much later in the routine.
: 953      1609 3      |
: 954      1610 P      | T =N ERR_MESSAGE (MOUN$ MOUNTED, 3, .LABEL_STRING[J+2],
: 955      1611      | .LABEL_STRING[J+2+1], -PHYS_NAME[J, LEN])
: 956      1612 3      |
: 957      1613 3      | ELSE
: 958      1614      | IF .STATUS GTRU 7
: 959      1615      | THEN ERR_EXIT (.STATUS)
: 960      1616      | ELSE
: 961      1617 3      |
: 962      1618 3      |
: 963      1619 3      | Volume not found: either not there or this is a /NOSHARE mount.
: 964      1620 3      | We must go through the mechanics of mounting the device.
: 965      1621 3      |
: 966      1622 4      | BEGIN
: 967      1623 4      |
: 968      1624 4      |
: 969      1625 4      | The following block of code should not be re-executed if this routine
: 970      1626 4      | is called a second time by operator-assisted mount code.
: 971      1627 4      |
: 972      1628 4      | IF NOT .DEV_ACQUIRED[J]
: 973      1629 4      | THEN
: 974      1630 5      | BEGIN
: 975      1631 5      |
: 976      1632 5      | LOCAL
: 977      1633 5      | STSBLK : VECTOR [2];
: 978      1634 5      |
: 979      1635 5      |
: 980      1636 5      | Call the SEARCH_DEVICE routine to search for a mountable device,
: 981      1637 5      | allocate it, and set up the physical device name and descriptor
: 982      1638 5      | in mount database. Note that if the device is available cluster-
: 983      1639 5      | wide, SEARCH_DEVICE will take out an EX mode lock for a private
: 984      1640 5      | mount, or a PW mode lock for a shared mount.
: 985      1641 5      |
: 986      1642 5      | IF NOT .DEV_ALLOCATED [J]
: 987      1643 5      | THEN
: 988      1644 6      | BEGIN
: 989      1645 6      | STATUS = KERNEL_CALL (SEARCH_DEVICE, .J);
: 990      1646 6      |
: 991      1647 6      |
: 992      1648 6      | If the device does not exists, disable operator assist before
: 993      1649 6      | exiting with the error status.
: 994      1650 6      |
: 995      1651 6      | Otherwise, indicate that this device has been allocated.
: 996      1652 6      | If the device was not previously allocated, indicate such.
: 997      1653 6      | If the mount fails, these devices must be deallocated.
: 998      1654 6      |
: 999      1655 6      | IF NOT .STATUS
: 1000     1656 6      | THEN
: 1001     1657 7      | BEGIN
: 1002     1658 8      | IF ((.STATUS AND STS$M_MSG_NO) EQL (SS$_NOSUCHDEV AND STS$M_MSG_NO))
: 1003     1659 8      | OR ((.STATUS AND STS$M_MSG_NO) EQL (SS$_IVDEVNAM AND STS$M_MSG_NO))
: 1004     1660 7      | THEN
: 1005     1661 7      | MOUNT_OPTIONS [OPT_ASSIST] = 0;
: 1006     1662 7      | ERR_EXIT (.STATUS);
: 1007     1663 6      | END;

```

```

1008      1664      6      IF .STATUS NEQ SSS_DEVALRALLOC
1009      1665      6      THEN
1010      1666      7          BEGIN
1011      1667      7              CLEANUP_FLAGS [CLF_DEALLOCATE] = 1;
1012      1668      7              CLEANUP_ALLOC [.J] = 1;
1013      1669      6              END;
1014      1670      6      DEV_ALLOCATED [.J] = 1;
1015      1671      5      END;
1016      1672      5
1017      1673      5
1018      1674      5      ! Set the PHYS_NAME high-water mark.
1019      1675      5
1020      1676      5      PHYS_COUNT = .J + 1;
1021      1677      5
1022      1678      5      END
1023      1679      5      ! End of code that shouldn't be executed more than once
1024      1680      5      ! per device.
1025      1681      4      ELSE
1026      1682      4
1027      1683      5      BEGIN
1028      1684      5
1029      1685      5      ! Take out a lock on the allocation class device name. This will
1030      1686      5      ! interlock all mounts of this device.
1031      1687      5
1032      1688      5      STATUS = $ENQW (LKMODE = LCK$K_EXMODE,
1033      1689      5              LKSB = LOCK_STATUS,
1034      1690      5              FLAGS = LCK$M_SYSTEM,
1035      1691      5              RESNAM = ALLDEVNAM_DESC,
1036      1692      5              EFN = MOUNT_EFN,
1037      1693      5              ACMODE = PSL$C_EXEC);
1038      1694      5      IF NOT .STATUS THEN ERR_EXIT (.STATUS);
1039      1695      5
1040      1696      4      END;
1041      1697      4
1042      1698      4
1043      1699      4      ! The remainder of the code is executed each time this routine is called by
1044      1700      4      ! ASSIST if an operator-assisted mount is required.
1045      1701      4
1046      1702      4      DEV_ACQUIRED[.J] = 1;
1047      1703      4
1048      1704      4
1049      1705      4      ! Get a channel to it. If this is a cluster accessible device,
1050      1706      4      ! a device lock will be taken out by this node on the device.
1051      1707      4
1052      1708      4
1053      1709      4      STATUS = $ASSIGN (DEVNAM = PHYS_NAME[.J,LEN],
1054      1710      4              CHAN = CHANNEL);
1055      1711      4      IF NOT .STATUS THEN ERR_EXIT (.STATUS);
1056      1712      4
1057      1713      4
1058      1714      4      ! Get the device characteristics and do device type validation: Make sure
1059      1715      4      ! the device is mountable at all, and check that the mount qualifiers are
1060      1716      4      ! consistent with the device type. A mismatch between primary and secondary
1061      1717      4      ! device characteristics indicates a spooled device or something else strange.
1062      1718      4      ! Reject such.
1063      1719      4
1064      1720      4

```

```

: 1065      1721  4 $GETCHN (CHAN = .CHANNEL, PRIBUF = DEVCHAR_DESC, SCDBUF = DEVCHAR_DESC2);
: 1066      1722  4
: 1067      1723  4 IF CH$NEQ (DIB$K_LENGTH, DEVICE_CHAR, DIB$K_LENGTH, DEVICE_CHAR2, 0)
: 1068      1724  4   OR NOT .DEVICE_CHAR[DEV$V_FOD]
: 1069      1725  4   THEN ERR_EXIT (SS$_NOTFILEDEV);
: 1070      1726  4
: 1071      1727  4 IF NOT .DEVICE_CHAR[DEV$V_AVL] THEN ERR_EXIT (SS$_DEVOFFLINE);
: 1072      1728  4
: 1073      1729  4 IF .DEVICE_CHAR[DEV$V_MNT] THEN ERR_EXIT (SS$_DEVMOUNT);
: 1074      1730  4
: 1075      1731  4 CLEANUP_FLAG$[CLF_CLEARVALID] = 1; ! device is now known not mounted
: 1076      1732  4
: 1077      1733  4 !
: 1078      1734  4 ! Some things to be tested on the 1st only and then stored anyway
: 1079      1735  4 !
: 1080      1736  4 IF .J EQL 0
: 1081      1737  4   THEN
: 1082      1738  5     BEGIN
: 1083      1739  5     !
: 1084      1740  5     ! is it a tape or disk mount
: 1085      1741  5     !
: 1086      1742  5     STORED_CONTEXT [TAPE_MOUNT] = .DEVICE_CHAR [DEV$V_SQD];
: 1087      1743  5     !
: 1088      1744  5     ! we need only to test if we are going to override something
: 1089      1745  5     ! once ( and then just save it )
: 1090      1746  5     !
: 1091      1747  6     IF ( .MOUNT_OPTIONS[OPT_FOREIGN] OR .MOUNT_OPTIONS[OPT_NO LABEL]
: 1092      1748  6     OR .MOUNT_OPTIONS[OPT_OVR_ACC] OR .MOUNT_OPTIONS[OPT_PROTECTION]
: 1093      1749  6     OR .MOUNT_OPTIONS[OPT_OVR_EXP] OR .MOUNT_OPTIONS[OPT_USER UIC]
: 1094      1750  6     OR .MOUNT_OPTIONS[OPT_NOQOTA] OR .MOUNT_OPTIONS[OPT_OWNER UIC]
: 1095      1751  6     OR .MOUNT_OPTIONS[OPT_OVR_LOCK] OR .MOUNT_OPTIONS[OPT_OVR_VOLO])
: 1096      1752  5     THEN STORED_CONTEXT [OVERRIDE_SOMETHING] = 1
: 1097      1753  5     ELSE STORED_CONTEXT [OVERRIDE_SOMETHING] = 0;
: 1098      1754  5     !
: 1099      1755  5     ! device number must match label number for disk
: 1100      1756  5     !
: 1101      1757  5     IF (NOT .STORED_CONTEXT [TAPE_MOUNT]) AND
: 1102      1758  6     (.DEVICE_COUNT NEQ .LABEL_COUNT) AND (.LABEL_COUNT NEQ 0)
: 1103      1759  5     THEN ERR_EXIT (MOUN$_DEVCOUNT);
: 1104      1760  5
: 1105      1761  4     END; ! End of block to be executed for first device only.
: 1106      1762  4
: 1107      1763  4 !
: 1108      1764  4 ! test legal options for device type
: 1109      1765  4 !
: 1110      1766  4 IF
: 1111      1767  5   BEGIN
: 1112      1768  5   IF .DEVICE_CHAR[DEV$V_SQD]
: 1113      1769  5   THEN
: 1114      1770  6     ((.OPTIONS[0] AND NOT TAPE_OPTIONS1) NEQ 0
: 1115      1771  6     OR (.OPTIONS[1] AND NOT TAPE_OPTIONS2) NEQ 0)
: 1116      1772  5   ELSE
: 1117      1773  6     ((.OPTIONS[0] AND NOT DISK_OPTIONS1) NEQ 0
: 1118      1774  6     OR (.OPTIONS[1] AND NOT DISK_OPTIONS2) NEQ 0)
: 1119      1775  5   END
: 1120      1776  4   THEN ERR_EXIT (MOUN$_ILLOPT);
: 1121      1777  4

```



```

1122 1778 4 ! device types must be consistent
1123 1779 4 ! tapes with tapes or disks with disks
1124 1780 4
1125 1781 5 IF (NOT .DEVICE_CHAR [DEV$V_SQD] AND .STORED_CONTEXT [TAPE_MOUNT])
1126 1782 4 OR
1127 1783 5 (.DEVICE_CHAR [DEV$V_SQD] AND NOT .STORED_CONTEXT [TAPE_MOUNT])
1128 1784 4 THEN ERR_EXIT (MOUN$_INCONSDEV);
1129 1785 4
1130 1786 4 !
1131 1787 4 ! Now attempt to read the home block or volume header label, as appropriate
1132 1788 4 ! for the device type.
1133 1789 4
1134 1790 4
1135 1791 4 IF .DEVICE_CHAR[DEV$V_SQD]
1136 1792 4 THEN
1137 1793 4 STATUS = READ_VOLLABEL (LABEL_STRING[J*2])
1138 1794 4 ELSE
1139 1795 4 STATUS = READ_HOMEBLOCK (LABEL_STRING[J*2], NOT .MOUNT_OPTIONS[OPT_FOREIGN]);
1140 1796 4
1141 1797 4 !
1142 1798 4 ! Now check the status of the volume against the various mount options. Note,
1143 1799 4 ! in particular, whether the user is attempting to override volume protection.
1144 1800 4 !
1145 1801 4
1146 1802 4 MOUNT_OPTIONS[OPT_IS_FILES11] = 1; ! assume volume is Files-11
1147 1803 4 IF NOT .STATUS
1148 1804 5 THEN BEGIN
1149 1805 5 IF .STATUS EQL SSS_NOHOMEBLK OR .STATUS EQL SSS_NOTLABELMT
1150 1806 5 ! if home block is not found
1151 1807 6 THEN BEGIN
1152 1808 6 MOUNT_OPTIONS[OPT_IS_FILES11] = 0;
1153 1809 7 IF NOT ( .MOUNT_OPTIONS[OPT_FOREIGN]
1154 1810 7 OR .MOUNT_OPTIONS[OPT_NOLABEL])
1155 1811 6 THEN
1156 1812 6 IF .DEVICE_CHAR[DEV$V_SQD]
1157 1813 6 THEN ERR_EXIT (.STATUS)
1158 1814 6 ELSE ERR_EXIT (.STATUS, 0, MOUN$ VOLIDENT, 6,
1159 1815 6 HM2$$_VOLNAME, HOME_BLOCK[HM2$T_VOLNAME],
1160 1816 6 HM2$$_OWNERNAME, HOME_BLOCK[HM2$T_OWNERNAME],
1161 1817 6 HM2$$_FORMAT, HOME_BLOCK[HM2$T_FORMAT]);
1162 1818 6 END
1163 1819 6
1164 1820 5 ELSE IF .STATUS EQL SSS_INCVOLLABEL ! if volume label mismatch
1165 1821 5 THEN
1166 1822 6 BEGIN
1167 1823 6 IF .MOUNT_OPTIONS[OPT_LABEL]
1168 1824 6 AND NOT .MOUNT_OPTIONS[OPT_FOREIGN]
1169 1825 6 AND NOT .MOUNT_OPTIONS[OPT_OVR_ID]
1170 1826 6 THEN
1171 1827 6 IF .DEVICE_CHAR[DEV$V_SQD]
1172 1828 6 THEN ERR_EXIT (.STATUS)
1173 1829 6 ELSE ERR_EXIT (.STATUS, 0, MOUN$ VOLIDENT, 6,
1174 1830 6 HM2$$_VOLNAME, HOME_BLOCK[HM2$T_VOLNAME],
1175 1831 6 HM2$$_OWNERNAME, HOME_BLOCK[HM2$T_OWNERNAME],
1176 1832 6 HM2$$_FORMAT, HOME_BLOCK[HM2$T_FORMAT]);
1177 1833 6 END
1178 1834 6

```

P
P
P

P
P
P

```
1179 1835 5 ELSE
1180 1836 6 BEGIN
1181 1837 6 MOUNT_OPTIONS[OPT_IS_FILES11] = 0; ! Clean up option flag.
1182 1838 6 ERR_EXIT (.STATUS);
1183 1839 5 END;
1184 1840 4 END;
1185 1841 4
1186 1842 4
1187 1843 4 !
1188 1844 4 ! are overriding something with a files-11 mount
1189 1845 4 IF .MOUNT_OPTIONS[OPT_IS_FILES11] AND .STORED_CONTEXT [OVERRIDE_SOMETHING]
1190 1846 4 THEN MOUNT_OPTIONS[OPT_OVR_PRO] = 1;
1191 1847 4
1192 1848 4 !
1193 1849 4 ! Call the device specific routine that actually does the mount.
1194 1850 4 !
1195 1851 4
1196 1852 4 IF .DEVICE_CHAR[DEV$V_SQD]
1197 1853 4 THEN
1198 1854 5 BEGIN
1199 1855 5 MOUNT_TAPE ();
1200 1856 5 KERNEL_CALL (XFER_DEV_OWNER, .CHANNEL);
1201 1857 5 END
1202 1858 4 ELSE
1203 1859 5 BEGIN
1204 1860 5
1205 1861 5 ! Get the device context, if it exists. This is necessary to
1206 1862 5 ! make sure that mounts of the same device from different nodes
1207 1863 5 ! are consistent.
1208 1864 5 !
1209 1865 5
1210 1866 6 IF NOT (STATUS = KERNEL_CALL (GET_DEVICE_CONTEXT))
1211 1867 5 THEN
1212 1868 5 ERR_EXIT (.STATUS);
1213 1869 5
1214 1870 5 IF .MOUNT_OPTIONS[OPT_IS_FILES11B]
1215 1871 5 THEN
1216 1872 5 MOUNT_DISK2 ()
1217 1873 5 ELSE
1218 1874 5 MOUNT_DISK1 ();
1219 1875 5 !
1220 1876 5 ! If we are mounting a shared volume on an allocated device, deallocate the
1221 1877 5 ! device now. We delayed the deallocation until now so that if the mount
1222 1878 5 ! failed, the device remained allocated.
1223 1879 5 !
1224 1880 5 IF NOT .MOUNT_OPTIONS [OPT_NOSHARE]
1225 1881 6 THEN KERNEL_CALL (DALLOC_SHR_DEV, .CHANNEL)
1226 1882 5 ELSE KERNEL_CALL (XFER_DEV_OWNER, .CHANNEL);
1227 1883 5
1228 1884 4 END;
1229 1885 4
1230 1886 4 !
1231 1887 4 ! Deassign the channel.
1232 1888 4 !
1233 1889 4 $DASSGN (CHAN = .CHANNEL);
1234 1890 4
1235 1891 3 END; ! shared mount path rejoins us here
```

```

: 1236 1892 2 END; ! end of rebind block
: 1237 1893 2
: 1238 1894 2
: 1239 1895 2 Clean out status values for the next time around the loop.
: 1240 1896 2
: 1241 1897 2
: 1242 1898 2 CLEANUP_FLAGS = .CLEANUP_FLAGS AND (1^CLF_REBUILD OR 1^CLF_REBUILDQUO);
: 1243 1899 2 CHANNEL = 0;
: 1244 1900 2 REAL_MVL = 0;
: 1245 1901 2 REAL_RVT = 0;
: 1246 1902 2 REAL_VCB = 0;
: 1247 1903 2 REAL_FCB = 0;
: 1248 1904 2 REAL_WCB = 0;
: 1249 1905 2 REAL_AQB = 0;
: 1250 1906 2 MTL_ENTRY = 0;
: 1251 1907 2 SMT_ENTRY = 0;
: 1252 1908 2 OPTIONS[0] = .OPTIONS[0] AND NOT RESET_OPTIONS1;
: 1253 1909 2 OPTIONS[1] = .OPTIONS[1] AND NOT RESET_OPTIONS2;
: 1254 1910 2 MOUNT_FAILED = 0; ! Indicate that the mount worked.
: 1255 1911 3 RETURN (SS$_NORMAL) ! Return success status
: 1256 1912 1 END; ! end of MOUNT_VOLUME

```

```

.PSECT SPLITS,NOWRT,NOEXE,2
001C 0000 P.AAA: .WORD 28
00EC 0002 .WORD 236
00000000' 00004 .ADDRESS ALLDEVNAM_BUF+4
00000000' 00008 .ADDRESS ALLDEVNAM_DESC
00000000 0000C .LONG 0
0004 00010 P.AAB: .WORD 4
10D0 00012 .WORD 4304
00000000 00014 .LONG 0
00000000 00018 .LONG 0
00000000 0001C .LONG 0

```

```

.EXTRN DEV_CTX, MOUNT_FAILED
.EXTRN LABEL_COUNT, DEVICE_STRING
.EXTRN LABEL_STRING, SEARCH_VOL
.EXTRN TRAN_COGNOME, READ_VOLLABEL
.EXTRN READ_HOMEBLOCK, MOUNT_TAPE
.EXTRN MOUNT_DISK1, MOUNT_DISK2
.EXTRN GET_DEVICE_CONTEXT
.EXTRN SYS$GETSYIQ, SYS$ENQW
.EXTRN SYS$GETCHN

```

```

.PSECT $CODE$,NOWRT,2

```

```

OFFC 0000 MOUNT_VOLUME:
SB 00000000G 00 9E 00002 .WORD Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11 : 1436
SA 00000000G 00 9E 00009 MCVAB LABEL_STRING, R11
S9 00000000G 00 9E 00010 MOVAB LIB$STOP, R10
S8 00000000' EF 9E 00017 MOVAB MOUNT_OPTIONS, R9
SE 24 C2 0001E SUBL2 #36, SP
14 AE 00000000' EF 10 28 00021 MOVCS #16, P.AAA, DEVICE_ITMLST1 : 1487

```

| | | | | | | | | | | | |
|------|----|-----------|----|------|----------|------|-------|--------|------------------------------|------------------------|------|
| | | 04 | 6D | 03BE | CF | DE | 0002A | MOVAL | 45\$, (FP) | | |
| | | 00000000G | A9 | 0207 | 8F | AA | 0002F | BICW2 | #51\$, OPTIONS+4 | | 1522 |
| | | | 00 | | 01 | DD | 00035 | MOVL | #1, MOUNT_FAILED | | 1523 |
| | 70 | | 69 | | 57 | D4 | 0003C | CLRL | STATUS | | 1555 |
| | | | | | 04 | E0 | 0003E | BBS | #4, MOUNT_OPTIONS, 1\$ | | 1556 |
| 04 | AE | 00000000' | EF | | 6E | D4 | 00042 | CLRL | CSID | | 1558 |
| | | 08 | AE | | 10 | 28 | 00044 | MOV3 | #16, P.AAB, SYI_ITMLST | | 1573 |
| | | | | | 6E | 9E | 0004D | MOVAB | CSID, SYI_ITMLST+4 | | 1558 |
| | | | | | 7E | 7C | 00051 | CLRQ | -(SP) | | 1576 |
| | | | | | 7E | D4 | 00053 | CLRL | -(SP) | | |
| | | | | 10 | AE | 9F | 00055 | PUSHAB | SYI_ITMLST | | |
| | | | | | 7E | 7C | 00058 | CLRQ | -(SP) | | |
| | | | | | 1A | DD | 0005A | PUSHL | #26 | | |
| | 50 | 00000000G | 00 | | 07 | FB | 0005C | CALLS | #7, SYS\$GETSYIW | | 1581 |
| | | 04 | AC | | 01 | 78 | 00063 | ASHL | #1, J, R0 | | |
| 018C | C8 | | 56 | | 6B40 | DE | 00068 | MOVAL | LABEL_STRING[R0], R6 | | |
| | | 016C | 66 | | 08 | C1 | 0006C | ADDL3 | #8, (R6), LABLCKNAM_DESC | | 1582 |
| | | | C8 | | 6E | DD | 00072 | MOVL | CSID, LABLCKNAM_BUF+4 | | 1584 |
| 0170 | C8 | | 50 | | 04 | AB40 | DD | 00077 | MOVL | LABEL_STRING+4[R0], R0 | 1587 |
| | | | 60 | | 66 | 28 | 0007C | MOV3 | (R6), -(R0), LABLCKNAM_BUF+8 | | 1594 |
| | | | 7E | | 01 | 7D | 00082 | MOVQ | #1, -(SP) | | |
| | | | | | 7E | 7C | 00085 | CLRQ | -(SP) | | |
| | | | | | 7E | 7C | 00087 | CLRQ | -(SP) | | |
| | | | | 018C | C8 | 9F | 00089 | PUSHAB | LABLCKNAM_DESC | | |
| | | | | | 10 | DD | 0008D | PUSHL | #16 | | |
| | | | | 0128 | C8 | 9F | 0008F | PUSHAB | LABLCK_STATUS | | |
| | | | | | 05 | DD | 00093 | PUSHL | #5 | | |
| | | | | | 1A | DD | 00095 | PUSHL | #26 | | |
| | | 00000000G | 00 | | 0B | FB | 00097 | CALLS | #11, SYS\$ENQW | | 1596 |
| | | | | | 56 | DD | 0009E | PUSHL | R6 | | |
| | | | | | 01 | DD | 000A0 | PUSHL | #1 | | |
| | | | | | 5E | DD | 000A2 | PUSHL | SP | | |
| | | | | | 0000G | CF | 9F | 000A4 | PUSHAB | SEARCH VOL | |
| | | 00000000G | 9F | | 04 | FB | 000A8 | CALLS | #4, @#SYS\$CMKRN | | |
| | | | 57 | | 50 | DD | 000AF | MOVL | R0, STATUS | | |
| | | | 28 | | 57 | E9 | 000B2 | BLBC | STATUS, 2\$ | | 1605 |
| | | | 50 | | 04 | AC | DD | 000B5 | MOVL | J, R0 | 1611 |
| | | | | | FB40 | C840 | 7F | 000B9 | PUSHAQ | PHYS_NAME[R0] | |
| | 51 | | 50 | | 01 | 78 | 000BE | ASHL | #1, R0, R1 | | |
| | | | | | 04 | AB41 | DD | 000C2 | PUSHL | LABEL_STRING+4[R1] | |
| | | | 50 | | 02 | C4 | 000C6 | MULL2 | #2, R0 | | |
| | | | | | 6B40 | DD | 000C9 | PUSHL | LABEL_STRING[R0] | | |
| | | | | | 03 | DD | 000CC | PUSHL | #3 | | |
| | | | | | 0072A003 | 8F | DD | 000CE | PUSHL | #7512067 | |
| | | 00000000G | 00 | | 05 | FB | 000D4 | CALLS | #5, LIB\$SIGNAL | | |
| | | | | | 0A | 11 | 000DB | BRB | 3\$ | | |
| | | | 07 | | 57 | D1 | 000DD | CMPL | STATUS, #7 | | 1614 |
| | | | | | 08 | 1B | 000E0 | BLEQU | 4\$ | | |
| | | | | | 57 | DD | 000E2 | PUSHL | STATUS | | 1615 |
| | | | 6A | | 01 | FB | 000E4 | CALLS | #1, LIB\$STOP | | |
| | | | | | 02D1 | 31 | 000E7 | BRW | 44\$ | | |
| | | | | | 04 | AC | DD | 000EA | MOVL | J, R4 | 1628 |
| | 64 | FB18 | C8 | | 54 | E0 | 000EE | BBS | R4, DEV_ACQUIRED, 10\$ | | |
| | 56 | FB14 | C8 | | 54 | E0 | 000F4 | BBS | R4, DEV_ALLOCATED, 9\$ | | 1642 |
| | | | | | 54 | DD | 000FA | PUSHL | R4 | | 1645 |
| | | | | | 01 | DD | 000FC | PUSHL | #1 | | |
| | | | | | 5E | DD | 000FE | PUSHL | SP | | |

| | | | | | | | | | |
|----|-----------|-----------|------|------|-------|--------|-------------------------|-------------------------|---------------------------------|
| | | 00000000V | EF | 9F | 00100 | PUSHAB | SEARCH_DEVICE | | |
| | 00000000G | 9F | 04 | FB | 00106 | CALLS | #4, @#SYSSCMKRNL | | |
| | | 57 | 50 | DO | 0010D | MOVL | R0, STATUS | | |
| | | 23 | 57 | E8 | 00110 | BLBS | STATUS, 7\$ | | 1655 |
| 50 | | 57 | 8F | CB | 00113 | BICL3 | #-65529, STATUS, R0 | | 1658 |
| | 00000908 | 8F | 50 | D1 | 0011B | CMPL | R0, #2312 | | |
| | | | 09 | 13 | 00122 | BEQL | 5\$ | | |
| | 00000140 | 8F | 50 | D1 | 00124 | CMPL | R0, #320 | | 1659 |
| | | | 04 | 12 | 0012B | BNEQ | 6\$ | | |
| | 06 | A9 | 04 | 8A | 0012D | BICB2 | #4, MOUNT_OPTIONS+6 | | 1661 |
| | | | 57 | DD | 00131 | PUSHL | STATUS | | 1662 |
| | | 6A | 01 | FB | 00133 | CALLS | #1, LIB\$STOP | | |
| | 00000641 | 8F | 57 | D1 | 00136 | CMPL | STATUS, #1601 | | 1664 |
| | | | 0B | 13 | 0013D | BEQL | 8\$ | | |
| | FB28 | C8 | 02 | 88 | 0013F | BISB2 | #2, CLEANUP_FLAGS | | 1667 |
| 00 | FB24 | C8 | 54 | E2 | 00144 | BBSS | R4, CLEANUP_ALLOC, 8\$ | | 1668 |
| 00 | FB14 | C8 | 54 | E2 | 0014A | BBSS | R4, DEV_ALLOCATED, 9\$ | | 1670 |
| | FB3C | C8 | 01 | A4 | 9E | 00150 | MOVAB | 1(R4), PHYS_COUNT | 1676 |
| | | | 27 | 11 | 00156 | BRB | 11\$ | | 1628 |
| | | 7E | 01 | 7D | 00158 | MOVQ | #1, -(SP) | | 1693 |
| | | | 7E | 7C | 0015B | CLRQ | -(SP) | | |
| | | | 7E | 7C | 0015D | CLRQ | -(SP) | | |
| | | 0150 | C8 | 9F | 0015F | PUSHAB | ALLDEVNAM_DESC | | |
| | | | 10 | DD | 00163 | PUSHL | #16 | | |
| | | FB1C | C8 | 9F | 00165 | PUSHAB | LOCK_STATUS | | |
| | | | 05 | DD | 00169 | PUSHL | #5 | | |
| | | | 1A | DD | 0016B | PUSHL | #26 | | |
| | 00000000G | 00 | 0B | FB | 0016D | CALLS | #11, SYSS\$ENQW | | |
| | | 57 | 50 | DO | 00174 | MOVL | R0, STATUS | | |
| | | 05 | 57 | E8 | 00177 | BLBS | STATUS, 11\$ | | 1694 |
| | | | 57 | DD | 0017A | PUSHL | STATUS | | |
| | | 6A | 01 | FB | 0017C | CALLS | #1, LIB\$STOP | | |
| 00 | FB18 | C8 | 54 | E2 | 0017F | BBSS | R4, DEV_ACQUIRED, 12\$ | | 1702 |
| | | | 7E | 7C | 00185 | CLRQ | -(SP) | | 1710 |
| | | FB2C | C8 | 9F | 00187 | PUSHAB | CHANNEL | | |
| | | FB40 | C844 | 7F | 0018B | PUSHAQ | PHYS_NAME[R4] | | |
| | 00000000G | 00 | 04 | FB | 00190 | CALLS | #4, SYSS\$ASSIGN | | |
| | | 57 | 50 | DO | 00197 | MOVL | R0, STATUS | | |
| | | 05 | 57 | E8 | 0019A | BLBS | STATUS, 13\$ | | 1711 |
| | | | 57 | DD | 0019D | PUSHL | STATUS | | |
| | | 6A | 01 | FB | 0019F | CALLS | #1, LIB\$STOP | | |
| | | | 0160 | C8 | 9F | 001A2 | PUSHAB | DEVCHAR_DESC2 | 1721 |
| | | | 7E | D4 | 001A6 | CLRL | -(SP) | | |
| | | | 0158 | C8 | 9F | 001A8 | PUSHAB | DEVCHAR_DESC | |
| | | | 7E | D4 | 001AC | CLRL | -(SP) | | |
| | | FB2C | C8 | DD | 001AE | PUSHL | CHANNEL | | |
| 74 | AB | 00000000G | 00 | 05 | FB | 001B2 | CALLS | #5, SYSS\$GETCHN | |
| | | | 68 | 0074 | 8F | 29 | 001B9 | CMPC3 | #116, DEVICE_CHAR, DEVICE_CHAR2 |
| | | | 05 | 12 | 001C0 | BNEQ | 14\$ | | 1723 |
| | 08 | 01 | AB | 06 | E0 | 001C2 | BBS | #6, DEVICE_CHAR+1, 15\$ | 1724 |
| | | | 7E | 01CC | 8F | 3C | 001C7 | MOVZWL | #460, -(SP) |
| | | | 6A | 01 | FB | 001CC | CALLS | #1, LIB\$STOP | 1725 |
| | 07 | 02 | AB | 02 | E0 | 001CF | BBS | #2, DEVICE_CHAR+2, 16\$ | 1727 |
| | | | 7E | 84 | 8F | 9A | 001D4 | MOVZBL | #132, -(SP) |
| | | | 6A | 01 | FB | 001D8 | CALLS | #1, LIB\$STOP | |
| 07 | 02 | AB | 03 | E1 | 001DB | BBC | #3, DEVICE_CHAR+2, 17\$ | | 1729 |
| | | 7E | 6C | 8F | 9A | 001E0 | MOVZBL | #108, -(SP) | |

| | | | | | | | |
|----|-----------|-----------|----------------|--------|---------------------------|------|--|
| | 000001DC | 8F | 09 13 002E2 | BEQL | 31\$ | | |
| | | | 57 D1 002E4 | CMPL | STATUS, #476 | | |
| | | | 10 12 002EB | BNEQ | 32\$ | | |
| | 04 | A9 | 02 8A 002ED | BICB2 | #2, MOUNT_OPTIONS+4 | 1808 | |
| 4C | 01 | A9 | 03 E0 002F1 | BBS | #3, MOUNT_OPTIONS+1, 36\$ | 1809 | |
| 47 | 01 | A9 | 04 E0 002F6 | BBS | #4, MOUNT_OPTIONS+1, 36\$ | 1810 | |
| | | | 18 11 002FB | BRB | 33\$ | 1812 | |
| | 0000010C | 8F | 57 D1 002FD | CMPL | STATUS, #268 | 1820 | |
| | | | 33 12 00304 | BNEQ | 34\$ | | |
| | | 03 | A9 95 00306 | TSTB | MOUNT_OPTIONS+3 | 1823 | |
| | | | 37 18 00309 | BGEQ | 36\$ | | |
| 32 | 01 | A9 | 03 E0 0030B | BBS | #3, MOUNT_OPTIONS+1, 36\$ | 1824 | |
| 2D | 02 | A9 | 06 E0 00310 | BBS | #6, MOUNT_OPTIONS+2, 36\$ | 1825 | |
| 24 | | 68 | 05 E0 00315 | BBS | #5, DEVICE_CHAR, 35\$ | 1827 | |
| | | | F0 A8 9F 00319 | PUSHAB | HOME_BLOCK+496 | 1832 | |
| | | | 0C DD 0031C | PUSHL | #12 | | |
| | | | E4 A8 9F 0031E | PUSHAB | HOME_BLOCK+484 | | |
| | | | 0C DD 00321 | PUSHL | #12 | | |
| | | | D8 A8 9F 00323 | PUSHAB | HOME_BLOCK+472 | | |
| | | | 0C DD 00326 | PUSHL | #12 | | |
| | | | 06 DD 00328 | PUSHL | #6 | | |
| | | 0072A00B | 8F DD 0032A | PUSHL | #7512075 | | |
| | | | 7E D4 00330 | CLRL | -(SP) | | |
| | | | 57 DD 00332 | PUSHL | STATUS | | |
| | | 6A | 0A FB 00334 | CALLS | #10, LIB\$STOP | 1827 | |
| | | | 09 11 00337 | BRB | 36\$ | 1837 | |
| | 04 | A9 | 02 8A 00339 | BICB2 | #2, MOUNT_OPTIONS+4 | 1838 | |
| | | | 57 DD 0033D | PUSHL | STATUS | | |
| | | 6A | 01 FB 0033F | CALLS | #1, LIB\$STOP | 1845 | |
| 0A | 04 | A9 | 01 E1 00342 | BBC | #1, MOUNT_OPTIONS+4, 37\$ | | |
| 04 | FBOC | C8 | 01 E1 00347 | BBC | #1, STORED_CONTEXT, 37\$ | | |
| | 04 | A9 | 01 88 0034D | BISB2 | #1, MOUNT_OPTIONS+4 | 1846 | |
| 07 | | 68 | 05 E1 00351 | BBC | #5, DEVICE_CHAR, 38\$ | 1852 | |
| | 0000G | CF | 00 FB 00355 | CALLS | #0, MOUNT_TAPE | 1855 | |
| | | | 3F 11 0035A | BRB | 42\$ | 1856 | |
| | | | 7E D4 0035C | CLRL | -(SP) | 1866 | |
| | | | 5E DD 0035E | PUSHL | SP | | |
| | | 0000G | CF 9F 00360 | PUSHAB | GET_DEVICE_CONTEXT | | |
| | 00000000G | 9F | 03 FB 00364 | CALLS | #3, @#SYSS\$CMKRNL | | |
| | | 57 | 50 D0 0036B | MOVL | R0, STATUS | | |
| | | 05 | 57 E8 0036E | BLBS | STATUS, 39\$ | | |
| | | | 57 DD 00371 | PUSHL | STATUS | 1868 | |
| | | 6A | 01 FB 00373 | CALLS | #1, LIB\$STOP | | |
| 07 | 04 | A9 | 02 E1 00376 | BBC | #2, MOUNT_OPTIONS+4, 40\$ | 1870 | |
| | 0000G | CF | 00 FB 0037B | CALLS | #0, MOUNT_DISK2 | 1872 | |
| | | | 05 11 00380 | BRB | 41\$ | | |
| | 0000G | CF | 00 FB 00382 | CALLS | #0, MOUNT_DISK1 | 1874 | |
| 10 | | 69 | 04 E0 00387 | BBS | #4, MOUNT_OPTIONS, 42\$ | 1880 | |
| | | | C8 DD 0038B | PUSHL | CHANNEL | 1881 | |
| | | FB2C | 01 DD 0038F | PUSHL | #1 | | |
| | | | 5E DD 00391 | PUSHL | SP | | |
| | | 00000000V | EF 9F 00393 | PUSHAB | DALLOC_SHR_DEV | | |
| | | | 0E 11 00399 | BRB | 43\$ | | |
| | | FB2C | C8 DD 0039B | PUSHL | CHANNEL | 1882 | |
| | | | 01 DD 0039F | PUSHL | #1 | | |
| | | | 5E DD 003A1 | PUSHL | SP | | |
| | | 00000000V | EF 9F 003A3 | PUSHAB | XFER_DEV_OWNER | | |

| | | | | | | | | | | |
|-----------|------|-----------|----|------|-------|-------|-------|-----------------------|---|------|
| 00000000G | 9F | | 04 | FB | 003A9 | 43\$: | CALLS | #4, @#SYSS\$CMKRNL | : | |
| | | FB2C | C8 | DD | 003B0 | | PUSHL | CHANNEL | : | 1889 |
| 00000000G | 00 | | 01 | FB | 003B4 | | CALLS | #1, SYSS\$DASSGN | : | |
| | FB2B | | C8 | CA | 003BB | 44\$: | BICL2 | #-1537, CLEANUP_FLAGS | : | 1898 |
| | | FB2C | C8 | D4 | 003C4 | | CLRL | CHANNEL | : | 1899 |
| | | 0100 | C8 | 7C | 003C8 | | CLRQ | REAL_MVL | : | 1900 |
| | | 0108 | C8 | D4 | 003CC | | CLRL | REAL_VCB | : | 1902 |
| | | 0110 | C8 | 7C | 003D0 | | CLRQ | REAL_FCB | : | 1903 |
| | | 0118 | C8 | 7C | 003D4 | | CLRQ | REAL_AQB | : | 1905 |
| | | 0120 | C8 | D4 | 003D8 | | CLRL | SMTL_ENTRY | : | 1907 |
| 04 | A9 | 0207 | 8F | AA | 003DC | | BICW2 | #519, OPTIONS+4 | : | 1909 |
| | | 00000000G | 00 | D4 | 003E2 | | CLRL | MOUNT_FAILED | : | 1910 |
| | 50 | | 01 | D0 | 003E8 | | MOVL | #1, RD | : | 1911 |
| | | | | 04 | 003EB | | RET | | : | 1912 |
| | | | | 0000 | 003EC | 45\$: | .WORD | Save nothing | : | 1487 |
| | | | | 7E | D4 | 003EE | CLRL | -(SP) | : | |
| | | | | 5E | DD | 003F0 | PUSHL | SP | : | |
| | 7E | 04 | AC | 7D | 003F2 | | MOVQ | 4(AP), -(SP) | : | |
| 00000000V | EF | | 03 | FB | 003F6 | | CALLS | #3, MAIN_HANDLER | : | |
| | | | | 04 | 003FD | | RET | | : | |

; Routine Size: 1022 bytes, Routine Base: \$CODE\$ + 0282


```

1258 1913 1 ROUTINE MAIN_HANDLER (SIGNAL, MECHANISM) =
1259 1914 1
1260 1915 1 :++
1261 1916 1
1262 1917 1 : FUNCTIONAL DESCRIPTION:
1263 1918 1
1264 1919 1 : This routine is the main level condition handler for the MOUNT
1265 1920 1 : utility. It undoes anything that MOUNT has done so far and returns
1266 1921 1 : the condition code as status to MOUNT's caller (i.e., the CLI).
1267 1922 1
1268 1923 1
1269 1924 1 : CALLING SEQUENCE:
1270 1925 1 : MAIN_HANDLER (ARG1, ARG2)
1271 1926 1
1272 1927 1 : INPUT PARAMETERS:
1273 1928 1 : ARu1: address of signal array
1274 1929 1 : ARG2: address of mechanism array
1275 1930 1
1276 1931 1 : IMPLICIT INPUTS:
1277 1932 1 : NONE
1278 1933 1
1279 1934 1 : OUTPUT PARAMETERS:
1280 1935 1 : NONE
1281 1936 1
1282 1937 1 : IMPLICIT OUTPUTS:
1283 1938 1 : NONE
1284 1939 1
1285 1940 1 : ROUTINE VALUE:
1286 1941 1 : NONE
1287 1942 1
1288 1943 1 : SIDE EFFECTS:
1289 1944 1 : stack unwound, control passed to CLI
1290 1945 1
1291 1946 1 :--
1292 1947 1
1293 1948 2 BEGIN
1294 1949 2
1295 1950 2 MAP
1296 1951 2 : SIGNAL : REF BBLOCK, ! signal array
1297 1952 2 : MECHANISM : REF BBLOCK; ! mechanism array
1298 1953 2
1299 1954 2 EXTERNAL
1300 1955 2 : USER_STATUS : VECTOR; ! status return of some routines
1301 1956 2
1302 1957 2
1303 1958 2 IF .SIGNAL[CHFSL_SIG_NAME] NEQ SSS_UNWIND
1304 1959 2 THEN
1305 1960 2 BEGIN
1306 1961 2 :
1307 1962 2 : Do cleanup as indicated by the status flags.
1308 1963 2 :
1309 1964 2 : IF .BBLOCK [SIGNAL[CHFSL_SIG_NAME], STSSV_SEVERITY] EQL STSSK_SEVERE
1310 1965 2 THEN
1311 1966 2 BEGIN
1312 1967 2 IF .CLEANUP_FLAGS[CLF_DISMOUNT]
1313 1968 2 THEN
1314 1969 2 KERNEL_CALL (FORCE_DISMOUNT);

```

```

1315 1970 4
1316 1971 4
1317 1972 4
1318 1973 5
1319 1974 5
1320 1975 5
1321 1976 5
1322 1977 6
1323 P 1978 6
1324 P 1979 6
1325 P 1980 6
1326 1981 6
1327 1982 6
1328 1983 6
1329 1984 5
1330 1985 5
1331 1986 5
1332 1987 4
1333 1988 4
1334 1989 4
1335 1990 4
1336 1991 4
1337 1992 4
1338 1993 4
1339 1994 4
1340 1995 4
1341 1996 3
1342 1997 2
1343 1998 2
1344 1999 2
1345 2000 2
1346 2001 2
1347 2002 2
1348 2003 1

```

```

IF .CHANNEL NEQ 0
THEN
BEGIN
IF NOT .CLEANUP_FLAGS[CLF_DISMOUNT]
AND .CLEANUP_FLAGS[CLF_CLEARVALID]
THEN
BEGIN
DO_IO (CHAN = .CHANNEL,
FUNC = (IOS_AVAILABLE OR IOSM_INHERLOG),
EFN = MOUNT_EFN);
IF .STORED_CONTEXT [TAPE MOUNT]
THEN KERNEL_CALL (CLEAR_VALID);
END;
SDASSGN (CHAN = .CHANNEL);
CHANNEL = 0;
END;

Zero the various cleanup flags.
CLEANUP_FLAGS[CLF_DISMOUNT] = 0;
CLEANUP_FLAGS[CLF_CLEARVALID] = 0;
CLEANUP_FLAGS[CLF_DEASSTEMP] = 0;

END;
END;

Resignal the condition. Does not affect UNWIND.
RETURN SSS_RESIGNAL;

END;
! end of routine MAIN_HANDLER

```

| | | | | .EXTRN COMMON_IO | | |
|----|----|----------|----|--------------------------|-----------------|------|
| | | | | 000C 00000 MAIN_HANDLER: | | |
| | | | 53 | 00000000G | 9F 9E 00002 | 1913 |
| | | | 52 | 00000000' | EF 9E 00009 | |
| | | | 50 | 04 | AC D0 00010 | 1958 |
| | | 00000920 | 8F | 04 | A0 D1 00014 | |
| | | | | | 6A 13 0001C | |
| 04 | 04 | A0 | 03 | | 00 E0 0001E | 1964 |
| | | | | | 62 12 00024 | |
| | | 0D | 62 | | 06 E1 00026 | 1967 |
| | | | | | 7E D4 0002A | 1969 |
| | | | | | 5E DD 0002C | |
| | | | 63 | 00000000V | EF 9F 0002E | |
| | | | | | 03 FB 00034 | |
| | | | | 04 | A2 D5 00037 18: | 1971 |
| | | | | | 41 13 0003A | |
| | | 30 | 62 | | 06 E0 0003C | 1974 |
| | | | | | 62 95 00040 | 1975 |

| | | | | | | | | | |
|-----------|----|-----------|----|-------|-------|--------|---------------------|---|------|
| | | | 2C | 18 | 00042 | BGEQ | 2\$ | : | |
| | | | 7E | 7C | 00044 | CLRQ | -(SP) | : | 1981 |
| | | | 7E | 7C | 00046 | CLRQ | -(SP) | : | |
| | | | 7E | 7C | 00048 | CLRQ | -(SP) | : | |
| | | | 7E | 7C | 0004A | CLRQ | -(SP) | : | |
| | | | 7E | D4 | 0004C | CLRL | -(SP) | : | |
| | 7E | 0811 | 8F | 3C | 0004E | MOVZWL | #2065, -(SP) | : | |
| | | 04 | A2 | DD | 00053 | PUSHL | CHANNEL | : | |
| | | | 1A | DD | 00056 | PUSHL | #26 | : | |
| 00000000G | 00 | | 0C | FB | 00058 | CALLS | #12, COMMON IO | : | |
| | 0D | E4 | A2 | E9 | 0005F | BLBC | STORED_CONTEXT, 2\$ | : | 1982 |
| | | | 7E | D4 | 00063 | CLRL | -(SP) | : | 1983 |
| | | | 5E | DD | 00065 | PUSHL | SP | : | |
| | 63 | 00000000V | EF | 9F | 00067 | PUSHAB | CLEAR_VALID | : | |
| | | | 03 | FB | 0006D | CALLS | #3, SYSSCMKRN | : | |
| | | | A2 | D4 | 00070 | PUSHL | CHANNEL | : | 1985 |
| 00000000G | 00 | | 01 | FB | 00073 | CALLS | #1, SYSSDASSGN | : | |
| | | | A2 | D4 | 0007A | CLRL | CHANNEL | : | 1986 |
| | 62 | 40 | 8F | 8A | 0007D | BICB2 | #64, CLEANUP_FLAGS | : | 1992 |
| | 62 | 80 | 8F | 8A | 00081 | BICB2 | #128, CLEANUP_FLAGS | : | 1993 |
| | 62 | | 10 | 8A | 00085 | BICB2 | #16, CLEANUP_FLAGS | : | 1994 |
| | 50 | 0918 | 8F | 3C | 00088 | MOVZWL | #2328, R0 | : | 2001 |
| | | | 04 | 0008D | RET | | | : | 2003 |

; Routine Size: 142 bytes, Routine Base: \$CODE\$ + 0680

```

1350 2004 1 ROUTINE FORCE_DISMOUNT =
1351 2005 1
1352 2006 1 ++
1353 2007 1
1354 2008 1 FUNCTIONAL DESCRIPTION:
1355 2009 1
1356 2010 1 This routine initiates a dismount on the volume just mounted
1357 2011 1 (usually because an error occurred during the /BIND processing).
1358 2012 1 This routine must be called in kernel mode.
1359 2013 1
1360 2014 1
1361 2015 1 CALLING SEQUENCE:
1362 2016 1 FORCE_DISMOUNT ()
1363 2017 1
1364 2018 1 INPUT PARAMETERS:
1365 2019 1 NONE
1366 2020 1
1367 2021 1 IMPLICIT INPUTS:
1368 2022 1 MTL_ENTRY: address of mounted volume list entry just created
1369 2023 1 SMT_ENTRY: as above, for volume set if non-zero
1370 2024 1
1371 2025 1 OUTPUT PARAMETERS:
1372 2026 1 NONE
1373 2027 1
1374 2028 1 IMPLICIT OUTPUTS:
1375 2029 1 NONE
1376 2030 1
1377 2031 1 ROUTINE VALUE:
1378 2032 1 1
1379 2033 1
1380 2034 1 SIDE EFFECTS:
1381 2035 1 volume dismounted
1382 2036 1
1383 2037 1 --
1384 2038 1
1385 2039 2 BEGIN
1386 2040 2
1387 2041 2 BUILTIN
1388 2042 2 REMQUE;
1389 2043 2
1390 2044 2 LINKAGE
1391 2045 2 IOC_DISMOUNT = JSB (REGISTER = 6, REGISTER = 3, REGISTER = 4) :
1392 2046 2 NOPRESERVE (2);
1393 2047 2
1394 2048 2 EXTERNAL
1395 2049 2 SCH$GL_CURPCB : REF BBLOCK ADDRESSING_MODE (ABSOLUTE);
1396 2050 2 ! address of process PCB
1397 2051 2
1398 2052 2 EXTERNAL ROUTINE
1399 2053 2 IOC$DISMOUNT : IOC_DISMOUNT ADDRESSING_MODE (GENERAL);
1400 2054 2 ! system dismount routine
1401 2055 2
1402 2056 2 LOCAL
1403 2057 2 MTL : REF BBLOCK; ! address of mount list entry
1404 2058 2
1405 2059 2
1406 2060 2 REMQUE (.MTL_ENTRY, MTL);

```

```

: 1407      2061 2 IOC$DISMOUNT (.MTL, 1, .SCH$GL_CURPCB);
: 1408      2062
: 1409      2063 IF .SMTL_ENTRY NEQ 0
: 1410      2064 THEN
: 1411      2065     BEGIN
: 1412      2066     REMQUE (.SMTL_ENTRY, MTL);
: 1413      2067     IOC$DISMOUNT (.MTL, 1, .SCH$GL_CURPCB);
: 1414      2068     END;
: 1415      2069
: 1416      2070 RETURN 1;
: 1417      2071
: 1418      2072 1 END;

```

. end of routine FORCE_DISMOUNT

.EXTRN SCH\$GL_CURPCB, IOC\$DISMOUNT

OFFC 00000 FORCE_DISMOUNT:

| | | | | | | | |
|----|-----------|----|----|-------|--------|--------------------------------------|--------|
| 57 | 00000000G | 00 | 9E | 00002 | .WORD | Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11 | : 2004 |
| 55 | 00000000G | 9F | 9E | 00009 | MOVAB | IOC\$DISMOUNT, R7 | |
| 56 | 00000000' | FF | 0F | 00010 | MOVAB | @SCH\$GL_CURPCB, R5 | |
| 54 | | 65 | D0 | 00017 | REMQUE | @MTL_ENTRY, MTL | : 2060 |
| 53 | | 01 | D0 | 0001A | MOVL | SCH\$GL_CURPCB, R4 | : 2061 |
| | | 67 | 16 | 0001D | MOVL | #1, R3 | |
| 50 | 00000000' | EF | D0 | 0001F | JSB | IOC\$DISMOUNT | |
| | | 0B | 13 | 00026 | MOVL | SMTL_ENTRY, R0 | : 2063 |
| 56 | | 60 | 0F | 00028 | BEQL | 1\$ | |
| 54 | | 65 | D0 | 0002B | REMQUE | (R0), MTL | : 2066 |
| 53 | | 01 | D0 | 0002E | MOVL | SCH\$GL_CURPCB, R4 | : 2067 |
| | | 67 | 16 | 00031 | MOVL | #1, R3 | |
| 50 | | 01 | D0 | 00033 | JSB | IOC\$DISMOUNT | : 2070 |
| | | 04 | 00 | 00036 | MOVL | #1, R0 | : 2072 |
| | | | | | RET | | |

: Routine Size: 55 bytes, Routine Base: \$CODE\$ + 070E

: 1419 2073 1

```

1421 2074 1 ROUTINE CLEAR_VALID =
1422 2075 1
1423 2076 1 !++
1424 2077 1
1425 2078 1 FUNCTIONAL DESCRIPTION:
1426 2079 1
1427 2080 1     This routine clears the volume valid bit in the UCB.
1428 2081 1
1429 2082 1
1430 2083 1 CALLING SEQUENCE:
1431 2084 1     CLEAR_VALID ( )
1432 2085 1
1433 2086 1 INPUT PARAMETERS:
1434 2087 1     NONE
1435 2088 1
1436 2089 1 IMPLICIT INPUTS:
1437 2090 1     CHANNEL: channel number assigned to device
1438 2091 1
1439 2092 1 OUTPUT PARAMETERS:
1440 2093 1     NONE
1441 2094 1
1442 2095 1 IMPLICIT OUTPUTS:
1443 2096 1     NONE
1444 2097 1
1445 2098 1 ROUTINE VALUE:
1446 2099 1     1
1447 2100 1
1448 2101 1 SIDE EFFECTS:
1449 2102 1     valid bit clear in UCB
1450 2103 1
1451 2104 1 !--
1452 2105 1
1453 2106 2 BEGIN
1454 2107 2
1455 2108 2 LOCAL
1456 2109 2     UCB           : REF BBLOCK;   ! pointer to UCB
1457 2110 2
1458 2111 2 EXTERNAL
1459 2112 2     CHANNEL;       ! channel assigned to device
1460 2113 2
1461 2114 2 EXTERNAL ROUTINE
1462 2115 2     GET_CHANNELUCB; ! get UCB of channel
1463 2116 2
1464 2117 2
1465 2118 2 ! Get the UCB address from the channel and clear the bit.
1466 2119 2 !
1467 2120 2
1468 2121 2 UCB = GET_CHANNELUCB (.CHANNEL);
1469 2122 2 UCB[UCBSV_VALID] = 0;
1470 2123 2
1471 2124 2 RETURN 1;
1472 2125 2
1473 2126 1 END;                                     ! end of routine CLEAR_VALID

```

.EXTRN GET_CHANNELUCB

| | | | | | | | | |
|-------|----|-------|------|----------|--------------|-------|--------------------|--------|
| | | | 0000 | 00000 | CLEAR_VALID: | | | |
| | | | | | | .WORD | Save nothing | : 2074 |
| | | | | | | PUSHL | CHANNEL | : 2121 |
| 0000G | CF | 0000G | CF | DD 00002 | | CALLS | #1, GET_CHANNELUCB | : 2122 |
| 65 | A0 | | 01 | FB 00006 | | BICB2 | #8, 1017UCB) | : 2124 |
| | 50 | | 08 | 8A 0000B | | MOVL | #1, R0 | : 2126 |
| | | | 01 | D0 0000F | | RET | | |
| | | | | 04 00012 | | | | |

; Routine Size: 19 bytes, Routine Base: \$CODE\$ + 0745

.....

```

1475 2127 1 GLOBAL ROUTINE DALLOC_SHR_DEV (CHANNEL) =
1476 2128 1
1477 2129 1 +-
1478 2130 1
1479 2131 1 FUNCTIONAL DESCRIPTION:
1480 2132 1
1481 2133 1 This routine locates the UCB associated with the channel passed to
1482 2134 1 it as an input argument. It then deallocates the device (i.e. marks
1483 2135 1 the UCB as unallocated) on the local system. If an
1484 2136 1 exclusive cluster-wide lock exists for this device, it will also
1485 2137 1 convert it into a CR mode lock.
1486 2138 1
1487 2139 1 CALLING SEQUENCE:
1488 2140 1
1489 2141 1 kernel_call (dalloc_shr_dev, .channel)
1490 2142 1
1491 2143 1 This routine must be called in kernel mode.
1492 2144 1
1493 2145 1 INPUT:
1494 2146 1
1495 2147 1 CHANNEL = channel to the device which is being mounted
1496 2148 1
1497 2149 1 OUTPUT:
1498 2150 1
1499 2151 1 None.
1500 2152 1
1501 2153 1 IMPLICIT INPUT:
1502 2154 1
1503 2155 1 Mount data base.
1504 2156 1 Device is being mounted /SHARE, /GROUP, or /SYSTEM.
1505 2157 1
1506 2158 1 IMPLICIT OUTPUT:
1507 2159 1
1508 2160 1 None.
1509 2161 1
1510 2162 1 ROUTINE VALUE:
1511 2163 1
1512 2164 1 1 if control is returned to the caller. Otherwise, the procedure
1513 2165 1 signals an error.
1514 2166 1
1515 2167 1 SIDE EFFECTS:
1516 2168 1
1517 2169 1 Device is deallocated. Device lock is converted to CR mode.
1518 2170 1 --
1519 2171 1
1520 2172 2 BEGIN
1521 2173 2
1522 2174 2 EXTERNAL ROUTINE
1523 2175 2 GET_CHANNELUCB: ADDRESSING_MODE (GENERAL);
1524 2176 2
1525 2177 2 LOCAL
1526 2178 2 STATUS, ! Status of $ENQ call.
1527 2179 2 LOCK_STATUS: VECTOR [2], ! Lock status block.
1528 2180 2 UCB: ^REF BBLOCK; ! UCB of device.
1529 2181 2
1530 2182 2 UCB = GET_CHANNELUCB (.CHANNEL); ! Get the UCB address.
1531 2183 2

```



```

1532 2184 2 : We already know that this is a shared mount; check to see if the device
1533 2185 2 : was previously allocated.
1534 2186 2 :
1535 2187 2 IF .UCB [UCB$$_PID] NEQ 0
1536 2188 2 THEN BEGIN
1537 2189 2 :
1538 2190 2 : Deallocate the local UCB.
1539 2191 2 :
1540 2192 2 UCB [UCB$$_PID] = 0;
1541 2193 2 BBLOCK [UCB [UCB$$_DEVCHAR], DEV$V_ALL] = 0;
1542 2194 2 UCB [UCB$$_REFC] = .UCB [UCB$$_REFC] - 1;
1543 2195 2 :
1544 2196 2 : If an exclusive lock exists, convert it to CR mode.
1545 2197 2 :
1546 2198 2 IF .UCB [UCB$$_LOCKID] NEQ 0
1547 2199 2 THEN BEGIN
1548 2200 2     LOCK STATUS [1] = .UCB [UCB$$_LOCKID];
1549 2201 2     STATUS = $ENQW (ACMODE = PSL$$_KERNEL,
1550 2202 2         EFN = MOUNT_EFN,
1551 2203 2         LKSB = LOCK STATUS,
1552 2204 2         FLAGS = (LCK$M_CONVERT OR LCK$M_CVTSYS),
1553 2205 2         LKMODE = LCK$$_CRMODE);
1554 2206 2     IF NOT .STATUS THEN ERR_EXIT (.STATUS);
1555 2207 2 END;
1556 2208 2 END;
1557 2209 2 :
1558 2210 2 RETURN (1);
1559 2211 2 :
1560 2212 2 END;

```

End of routine DALLOC_SHR_DEV.

| | | | | | | |
|-----------|----|----|-------------|--------|------------------------------|--------|
| | | | 0000 0000 | .ENTRY | DALLOC_SHR_DEV, Save nothing | : 2127 |
| | 5E | | 08 C2 00002 | SUBL2 | #8, SP | : 2182 |
| 00000000G | 00 | 04 | AC DD 00005 | PUSHL | CHANNEL | : 2187 |
| | | | 01 FB 00008 | CALLS | #1, GET_CHANNELUCB | |
| | | 2C | A0 D5 0000F | TSTL | 44(UCB) | : 2192 |
| | | | 3B 13 00012 | BEQL | 1\$ | : 2193 |
| | | 2C | A0 D4 00014 | CLRL | 44(UCB) | : 2194 |
| 3A | A0 | 80 | 8F 8A 00017 | BICB2 | #128, 58(UCB) | : 2198 |
| | | 5C | A0 B7 0001C | DECW | 92(UCB) | : 2200 |
| | | 20 | A0 D5 0001F | TSTL | 32(UCB) | : 2205 |
| | | | 2B 13 00022 | BEQL | 1\$ | |
| | 04 | AE | A0 D0 00024 | MOVL | 32(UCB), LOCK_STATUS+4 | |
| | | | 7E 7C 00029 | CLRQ | -(SP) | |
| | | | 7E 7C 0002B | CLRQ | -(SP) | |
| | | | 7E 7C 0002D | CLRQ | -(SP) | |
| | | | 7F D4 0002F | CLRL | -(SP) | |
| | 7E | 42 | 8F 9A 00031 | MOVZBL | #66, -(SP) | |
| | | 20 | AE 9F 00035 | PUSHAB | LOCK_STATUS | |
| | | | 01 DD 00038 | PUSHL | #1 | |
| | | | 1A DD 0003A | PUSHL | #26 | |
| 00000000G | 00 | | 0B FB 0003C | CALLS | #11, SYS\$ENQW | : 2206 |
| | 09 | | 50 EB 00043 | BLBS | STATUS, 1\$ | |
| | | | 50 DD 00046 | PUSHL | STATUS | |

VMOUNT
V04-002

L 7
16-Sep-1984 01:00:56
12-Sep-1984 11:14:53

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[MOUNT.SRC]VMOUNT.B32;3 (11) Page 48

VM
VO

00000000G 00
50

01 FB 00048
01 DO 0004F 1\$:
04 00052

CALLS #1, LIB\$STOP
MOVL #1, R0
RET

:
: 2210
: 2212

; Routine Size: 83 bytes, Routine Base: \$CODE\$ + 0758

```

1562 2213 1 GLOBAL ROUTINE XFER_DEV_OWNER (CHANNEL) =
1563 2214 1
1564 2215 1 +-
1565 2216 1
1566 2217 1 FUNCTIONAL DESCRIPTION:
1567 2218 1
1568 2219 1 This routine locates the UCB associated with the channel passed to
1569 2220 1 it as an input argument. If current process is a subprocess, then
1570 2221 1 the device ownership is transferred to the top level process in the
1571 2222 1 process tree. This is necessary to support job-wide mount.
1572 2223 1
1573 2224 1 Note: we perform the transfer of ownership simply by setting the
1574 2225 1 master's PID into the UCB. This is sufficient because the lock on
1575 2226 1 this device is not tied to this process, i.e. it is a system-owned
1576 2227 1 lock.
1577 2228 1
1578 2229 1 CALLING SEQUENCE:
1579 2230 1
1580 2231 1 KERNEL_CALL (XFER_DEV_OWNER, .CHANNEL)
1581 2232 1
1582 2233 1 This routine must be called in kernel mode.
1583 2234 1
1584 2235 1 INPUT:
1585 2236 1
1586 2237 1 CHANNEL = channel to the device which is being mounted
1587 2238 1
1588 2239 1 OUTPUT:
1589 2240 1
1590 2241 1 None.
1591 2242 1
1592 2243 1 IMPLICIT INPUT:
1593 2244 1
1594 2245 1 Mount data base.
1595 2246 1 Device is being mounted /NOSHARE.
1596 2247 1
1597 2248 1 IMPLICIT OUTPUT:
1598 2249 1
1599 2250 1 If the current process is a subprocess, then the device is
1600 2251 1 allocated to its master,
1601 2252 1 else
1602 2253 1 none.
1603 2254 1
1604 2255 1 ROUTINE VALUE:
1605 2256 1
1606 2257 1 1.
1607 2258 1
1608 2259 1 SIDE EFFECTS:
1609 2260 1
1610 2261 1 None.
1611 2262 1
1612 2263 1 --
1613 2264 1
1614 2265 2 BEGIN
1615 2266 2
1616 2267 2 EXTERNAL
1617 2268 2 SCH$GL_CURPCB : REF BBLOCK ADDRESSING_MODE (GENERAL);
1618 2269 2 ! address of our PCB

```

```

: 1619 2270 2
: 1620 2271 2 EXTERNAL ROUTINE
: 1621 2272 2 GET_CHANNELUCB : ADDRESSING_MODE (GENERAL);
: 1622 2273 2
: 1623 2274 2 LOCAL
: 1624 2275 2 JIB : REF BBLOCK, ! JIB of current process
: 1625 2276 2 UCB : REF BBLOCK ! UCB of device.
: 1626 2277 2
: 1627 2278 2 UCB = GET_CHANNELUCB (.CHANNEL); ! Get the UCB address.
: 1628 2279 2
: 1629 2280 2 ! We already know that this is a private mount; check to see if the device
: 1630 2281 2 ! was previously allocated for sanity's sake.
: 1631 2282 2
: 1632 2283 2 IF .UCB [UCB$$_PID] NEQ 0
: 1633 2284 2 THEN
: 1634 2285 2 BEGIN
: 1635 2286 2
: 1636 2287 2 ! Check if the current process is a subprocess. If so, set the PID
: 1637 2288 2 ! of the top level process in the p ocess tree in the UCB.
: 1638 2289 2
: 1639 2290 2 IF .SCH$GL_CURPCB [PCB$$_OWNER] NEQ 0
: 1640 2291 2 THEN
: 1641 2292 2 BEGIN
: 1642 2293 2 JIB = .SCH$GL_CURPCB [PCB$$_JIB];
: 1643 2294 2 UCB [UCB$$_PID] = .JIB [JIB$$_MPID];
: 1644 2295 2 END;
: 1645 2296 2 END;
: 1646 2297 2
: 1647 2298 2 RETURN 1;
: 1648 2299 2
: 1649 2300 1 END;

```

! End of routine XFER_DEV_OWNER.

| | | | | | | | |
|-----------|----|-----------|------|-------------|--------|------------------------------|--------|
| | | | 0000 | 00000 | .ENTRY | XFER_DEV_OWNER, Save nothing | : 2213 |
| | | 04 | AC | DD 00002 | PUSHL | CHANNEL | : 2278 |
| 00000000G | 00 | | 01 | FB 00005 | CALLS | #1, GET_CHANNELUCB | |
| | | 2C | A0 | D5 0000C | TSTL | 44(UCB) | : 2283 |
| | | | 16 | 13 0000F | BEQL | 1\$ | |
| | 51 | 00000000G | 00 | D0 00011 | MOVL | SCH\$GL_CURPCB, R1 | : 2290 |
| | | | 1C | A1 D5 00018 | TSTL | 28(R1) | |
| | | | 0A | 13 0001B | BEQL | 1\$ | |
| | 51 | 0080 | C1 | D0 0001D | MOVL | 128(R1), JIB | : 2293 |
| | 2C | A0 | 54 | A1 D0 00022 | MOVL | 84(JIB), 44(UCB) | : 2294 |
| | | 50 | 01 | D0 00027 | MOVL | #1, R0 | : 2298 |
| | | | 04 | 0002A | RET | | : 2300 |

: Routine Size: 43 bytes, Routine Base: \$CODE\$ + 07AB

: 1650 2301 1

```

1652 2302 1
1653 2303 1 GLOBAL ROUTINE MOUNT_CLUSTER (ITEM_LIST) =
1654 2304 1
1655 2305 1 |
1656 2306 1 |
1657 2307 1 | FUNCTIONAL DESCRIPTION:
1658 2308 1 |
1659 2309 1 |     This routine performs the cluster-wide mount operation.
1660 2310 1 |     It calls another routine to create a cluster-mount packet
1661 2311 1 |     and then sends this mount request to other nodes in the
1662 2312 1 |     cluster.
1663 2313 1 |
1664 2314 1 | CALLING SEQUENCE:
1665 2315 1 |
1666 2316 1 |     MOUNT_CLUSTER (ARG1)
1667 2317 1 |
1668 2318 1 | INPUTS:
1669 2319 1 |
1670 2320 1 |     ARG1      : Address of the mount item list
1671 2321 1 |
1672 2322 1 | OUTPUTS:
1673 2323 1 |
1674 2324 1 |     None.
1675 2325 1 |
1676 2326 1 | IMPLICIT INPUTS:
1677 2327 1 |
1678 2328 1 |     None.
1679 2329 1 |
1680 2330 1 | OUTPUT PARAMETERS:
1681 2331 1 |
1682 2332 1 |     None.
1683 2333 1 |
1684 2334 1 | IMPLICIT OUTPUTS:
1685 2335 1 |
1686 2336 1 |     None.
1687 2337 1 |
1688 2338 1 | ROUTINE VALUE:
1689 2339 1 |
1690 2340 1 |     1          : If success
1691 2341 1 |     Otherwise : Status from comm primitive.
1692 2342 1 |
1693 2343 1 | SIDE EFFECTS:
1694 2344 1 |
1695 2345 1 |     The mount request is sent to other nodes in the cluster.
1696 2346 1 |
1697 2347 1 | -
1698 2348 1 |
1699 2349 1 |
1700 2350 2 BEGIN                                ! Start of MOUNT_CLUSTER
1701 2351 2
1702 2352 2 MAP
1703 2353 2     ITEM_LIST      : REF BBLOCK;
1704 2354 2
1705 2355 2 EXTERNAL ROUTINE
1706 2356 2     IN_CLUSTER      : ADDRESSING_MODE (GENERAL),
1707 2357 2     SEND_CLUSTER    : ADDRESSING_MODE (GENERAL),
1708 2358 2     GET_OIC         : ADDRESSING_MODE (GENERAL);

```

```

1709 2359 2 EXTERNAL
1710 2360 2 MOUNT_OPTIONS : BITVECTOR VOLATILE; ! Parser option flags
1711 2361 2
1712 2362 2
1713 2363 2
1714 2364 2 ! Define constants to calculate the size of the cluster-mount buffer
1715 2365 2
1716 2366 2 LITERAL
1717 2367 2 ITEM_SIZE = 12,
1718 2368 2 NO_OF_ITEMS = 18,
1719 2369 2 BUFFER_SIZE = 63,
1720 2370 2 COMMENT_SIZE = 80,
1721 2371 2 ITEM_LIST_SIZE = ( (ITEM_SIZE*DEVMAX)*2 + (NO_OF_ITEMS*ITEM_SIZE) +4 );
1722 2372 2
1723 2373 2 LOCAL
1724 2374 2 STATUS,
1725 2375 2 LENGTH,
1726 2376 2 UIC;
1727 2377 2
1728 2378 2 OWN
1729 2379 2 BUFFER : VECTOR [0], ! Buffer area for
1730 2380 2 ! cluster-mount packet
1731 2381 2 ITEM_BUF : BBLOCK [ITEM_LIST_SIZE], ! Item descriptors
1732 2382 2
1733 2383 2 LABEL_BUF : BBLOCK [BUFFER_SIZE * DEVMAX], ! Volume labels
1734 2384 2 LOGNAM_BUF : BBLOCK [BUFFER_SIZE * DEVMAX], ! Logical names
1735 2385 2 ACPNAM_BUF : BBLOCK [BUFFER_SIZE], ! ACP name
1736 2386 2 VOLSET_BUF : BBLOCK [BUFFER_SIZE], ! Volume set name
1737 2387 2 COMMENT_BUF : BBLOCK [COMMENT_SIZE], ! Comments
1738 2388 2
1739 2389 2 NAME_BUF : VECTOR [NAMEBUF_LEN * DEVMAX, BYTE], ! Device names
1740 2390 2 BUFFER_END : VECTOR [0];
1741 2391 2
1742 2392 2 LITERAL
1743 2393 2 BUFFER_LEN = BUFFER_END - BUFFER;
1744 2394 2
1745 2395 2
1746 2396 2 IF ( NOT .MOUNT_OPTIONS [OPT_CLUSTER] ) ! If not /cluster or not in a
1747 2397 2 OR NOT ( STATUS = IN_CLUSTER ) ! cluster environment, return
1748 2398 2 THEN ! immediately
1749 2399 2 RETURN 1;
1750 2400 2
1751 2401 2 CH$FILL (0, BUFFER_LEN, BUFFER); ! Zero buffer area
1752 2402 2 STATUS = MOUNT_ENCIPHER (.ITEM_LIST, BUFFER_LEN, BUFFER, LENGTH);
1753 2403 2 ! Encipher the mount request
1754 2404 2 IF NOT .STATUS ! If error, return
1755 2405 2 THEN
1756 2406 2 RETURN .STATUS;
1757 2407 2
1758 2408 2 UIC = KERNEL CALL (GET UIC); ! Get our UIC
1759 2409 2 STATUS = KERNEL_CALL (SEND_CLUSTER, BUFFER, .LENGTH, UIC); ! Broadcast the request
1760 2410 2 ! Arg3 (UIC) non-zero means a cluster-mount
1761 2411 2 RETURN .STATUS;
1762 2412 2
1763 2413 2 1 END; ! End of MOUNT_CLUSTER

```

```

.PSECT $OWNS,NOEXE,2
00000 BUFFER: .BLKB 0
00000 ITEM_BUF: .BLKB 604
0025C LABEL_BUF: .BLKB 1008
0064C LOGNAM_BUF: .BLKB 1008
00A3C ACPNAM_BUF: .BLKB 63
00A7B .BLKB 1
00A7C VOLSET_BUF: .BLKB 63
00ABB .BLKB 1
00ABC COMMENT_BUF: .BLKB 80
00B0C NAME_BUF: .BLKB 512
00DOC BUFFER_END: .BLKB 0

```

```

.EXTRN IN_CLUSTER, SEND_CLUSTER
.EXTRN GET_UIC

```

```

.PSECT $CODE$,NOWRT,2

```

```

.ENTRY MOUNT_CLUSTER, Save R2,R3,R4,R5,R6,R7,R8

```

```

MOVAB @SYS$CMKRNL, R8 : 2303
MOVAB BUFFER, R7
SUBL2 #4, SP
BBC #6, MOUNT_OPTIONS+7, 1$ : 2396
CALLS #0, IN_CLUSTER : 2397
MOVL R0, STATUS
BLBS STATUS, 2$
MOVL #1, R0 : 2399
RET
MOVCS #0, (SP), #0, #3340, BUFFER : 2401
PUSHR #^M<R7, SP> : 2402
MOVZWL #3340, -(SP)
PUSHL ITEM_LIST
CALLS #4, MOUNT_ENCIPHER
MOVL R0, STATUS
BLBC STATUS, 3$ : 2404
CLRL -(SP) : 2408
PUSHL SP
PUSHAB GET_UIC
CALLS #3, SYS$CMKRNL : 2409
PUSHL UIC
PUSHL LENGTH
PUSHL R7
PUSHL #3
PUSHL SP
PUSHAB SEND_CLUSTER
CALLS #6, SYS$CMKRNL

```

```

01FC 00000
58 00000000G 9F 9E 00002
57 00000000' EF 9E 00009
5E 04 C2 00010
OD 0000G CF 06 E1 00013
00000000G 00 00 FB 00019
56 50 D0 00020
04 56 E8 00023
50 01 D0 00026 1$:
00 04 00029
6E 00 2C 0002A 2$:
4080 8F BB 00032
ODOC 8F 3C 00036
04 AC DD 0003B
00000000V EF 04 FB 0003E
56 50 D0 00045
24 56 E9 00048
7E D4 0004B
5E DD 0004D
00000000G 00 9F 0004F
68 03 FB 00055
04 AE DD 0005A
57 DD 0005D
03 DD 0005F
5E DD 00061
00000000G 00 9F 00063
68 06 FB 00069

```

VMOUNT
V04-002

E 8
12-Sep-1984 01:00:56
12-Sep-1984 11:14:53

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[MOUNT.SRC]VMOUNT.B32;3 (13) Page 54

56
50

50 DO 0006C
56 DO 0006F 3\$:
04 00072

MOVL RO, STATUS
MOVL STATUS, RO
RET

: 2411
: 2413

; Routine Size: 115 bytes. Routine Base: \$CODE\$ + 07D6

: 1764 2414 1
: 1765 2415 1

VM
VO


```

1767 2416 1
1768 2417 1 ROUTINE MOUNT_ENCIPHER (ITEM_LIST, LIMIT, BUFFER, LENGTH) =
1769 2418 1
1770 2419 1
1771 2420 1
1772 2421 1
1773 2422 1
1774 2423 1
1775 2424 1
1776 2425 1
1777 2426 1
1778 2427 1
1779 2428 1
1790 2429 1
1791 2430 1
1792 2431 1
1793 2432 1
1794 2433 1
1795 2434 1
1796 2435 1
1797 2436 1
1798 2437 1
1799 2438 1
1800 2439 1
1801 2440 1
1802 2441 1
1803 2442 1
1804 2443 1
1805 2444 1
1806 2445 1
1807 2446 1
1808 2447 1
1809 2448 1
1810 2449 1
1811 2450 1
1812 2451 1
1813 2452 1
1814 2453 1
1815 2454 1
1816 2455 1
1817 2456 1
1818 2457 1
1819 2458 1
1820 2459 1
1821 2460 1
1822 2461 1
1823 2462 1

```

ROUTINE MOUNT_ENCIPHER (ITEM_LIST, LIMIT, BUFFER, LENGTH) =

↑

FUNCTIONAL DESCRIPTION:

This routine takes the parameters of the mount request and enciphers the parameters into a cluster-mount packet.

CALLING SEQUENCE:

MOUNT_ENCIPHER (ARG1,ARG2,ARG3,ARG4)

INPUTS:

ARG1 : Address of the item list
ARG2 : Output buffer limit

OUTPUTS:

None.

IMPLICIT INPUTS:

None.

OUTPUT PARAMETERS:

ARG3 : Address of the output buffer to receive the cluster-mount packet
ARG4 : Address of a longword to receive the length of the output buffer

IMPLICIT OUTPUTS:

None.

ROUTINE VALUES:

1 : If successful
SS\$_BUFFEROVF : Insufficient internal buffer space, i.e. length exceeds limit

SIDE EFFECTS:

None.

NOTES:

This encipher routine takes the given mount item list and turns it into a cluster-mount packet of the form:

```

      Offset
      ↑-----↑
      : code1 : len1 : 0 ITEM LENG item_desc_1

```

```

1824 2473 1  |-----|
1825 2474 1  |offset to str_1| 8  ITEM_ADDR
1826 2475 1  |-----|
1827 2476 1  |      unused   | 12 ITEM_NULL
1828 2477 1  |-----|
1829 2478 1  |code2 : len2  | 0  ITEM LENG item_desc_2
1830 2479 1  |-----|
1831 2480 1  |offset to str_2| 8  ITEM_ADDR
1832 2481 1  |-----|
1833 2482 1  |      unused   | 12 ITEM_NULL
1834 2483 1  |-----|
1835 2484 1  |      .        |
1836 2485 1  |      .        |
1837 2486 1  |-----|
1838 2487 1  |      .        |
1839 2488 1  |      0        |      End of item decsptors
1840 2489 1  |-----|
1841 2490 1  |      str_1    |
1842 2491 1  |-----|
1843 2492 1  |      .....   |
1844 2493 1  |-----|
1845 2494 1  |      str_2    |
1846 2495 1  |-----|
1847 2496 1  |      .....   |
1848 2497 1  |-----|
1849 2498 1  |
1850 2499 1  |
1851 2500 1  |
1852 2501 1  |
1853 2502 1  |
1854 2503 1  |
1855 2504 1  |
1856 2505 1  |
1857 2506 1  |
1858 2507 1  |
1859 2508 1  |
1860 2509 1  |
1861 2510 1  |
1862 2511 2  | BEGIN                ! Start of MOUNT_ENCIPHER
1863 2512 2  |
1864 2513 2  | MAP
1865 2514 2  |     ITEM LIST       : REF BBLOCK,
1866 2515 2  |     BUFFER          : REF BBLOCK;
1867 2516 2  |
1868 2517 2  | LOCAL
1869 2518 2  |     ITEM            : REF BBLOCK,      ! Pointer to item descriptor
1870 2519 2  |     PTR             : REF BBLOCK,      ! Pointer to output item desc
1871 2520 2  |     STR PIR        : REF BBLOCK,      ! Pointer to item string
1872 2521 2  |     ITEM COUNT,    :                   ! Number of items in item list
1873 2522 2  |     DEVICE_COUNT,  :                   ! Device number index
1874 2523 2  |     J;
1875 2524 2  |
1876 2525 2  | EXTERNAL
1877 2526 2  |     MOUNT_OPTIONS  : BITVECTOR VOLATILE; ! Parser option flags
1878 2527 2  |
1879 2528 2  | MACRO ITEM LENG    = 0,0,16,0%;      ! Define buffer offsets
1880 2529 2  | MACRO ITEM CODE    = 2,0,16,0%;

```

1. This cluster-mount packet is to be sent to other nodes in the cluster and processed by CSP (the Cluster Server Process).
2. The address in the item descriptor is "relocated" to be the offset from the beginning of the packet (i.e. self-relative). This is done so that when CSP gets the cluster-mount packet, it can simply add the packet address to the offset in the item descriptor to obtain the address of the string.

```

1881 2530 2 MACRO ITEM_ADDR = 4,0,32,0%;
1882 2531 2 MACRO ITEM_NULL = 8,0,32,0%;
1883 2532 2 LITERAL ITEM_SIZE = 12;
1884 2533 2
1885 2534 2
1886 2535 2 : Count of number of items in the item list
1887 2536 2
1888 2537 2
1889 2538 2 ITEM = .ITEM_LIST; : Point to the beginning of list
1890 2539 2 ITEM_COUNT = 0; : Initialize counter
1891 2540 2 WHILE ( .ITEM [ITEM_CODE] NEQ 0 ) DO
1892 2541 2 BEGIN
1893 2542 2     ITEM_COUNT = .ITEM_COUNT + 1; : Increment number of items
1894 2543 2     ITEM = .ITEM + ITEM_SIZE; : Bump item descriptor pointer
1895 2544 2 END;
1896 2545 2
1897 2546 2
1898 2547 2 : Calculate space needed for the item descriptors
1899 2548 2
1900 2549 2 STR_PTR = .BUFFER + (.ITEM_COUNT * ITEM_SIZE); : Space req for descriptors
1901 2550 2 STR_PTR [ITEM_CODE] = STR_PTR [ITEM_LEN] = 0; : Mark end of descriptor area
1902 2551 2 STR_PTR = .STR_PTR + 4; : Mark beginning of string area
1903 2552 2 PTR = .BUFFER; : Mark beginning of descriptor area
1904 2553 2 ITEM = .ITEM_LIST; : Point to the beginning of item list
1905 2554 2 DEVICE_COUNT = 0; : Initialize device index
1906 2555 2 .LENGTH = 4; : Initialize length (itmlst stopper)
1907 2556 2
1908 2557 2
1909 2558 2 : For each item in the item list, copy the item descriptor and the
1910 2559 2 : item string to the output buffer
1911 2560 2
1912 2561 2 DECR J FROM .ITEM_COUNT TO 1 DO
1913 2562 2 BEGIN
1914 2563 2     SELECT .ITEM [ITEM_CODE] OF
1915 2564 2     SET
1916 2565 2     [MNT$DEVNAM] :
1917 2566 2     :
1918 2567 2     : BEGIN
1919 2568 2     : For DEVNAM:
1920 2569 2     :
1921 2570 2     : a. Create item descriptor, relocate address
1922 2571 2     : b. Compute length, return SSS_BUFFEROVF if appropriate
1923 2572 2     : c. Copy device string from physical device descriptor
1924 2573 2     :
1925 2574 2     :
1926 2575 2     : BIND
1927 2576 2     : DEV_DSC = PHYS_NAME [.DEVICE_COUNT * 2] : $BLOCK;
1928 2577 2     :
1929 2578 2     : PTR [ITEM_LEN] = .DEV_DSC [DSC$W_LENGTH];
1930 2579 2     : PTR [ITEM_CODE] = MNT$DEVNAM;
1931 2580 2     : PTR [ITEM_ADDR] = .STR_PTR - .BUFFER;
1932 2581 2     : .LENGTH = .LENGTH + ITEM_SIZE + .PTR [ITEM_LEN];
1933 2582 2     : IF .LENGTH GTRU .LIMIT
1934 2583 2     : THEN
1935 2584 2     :     RETURN SSS_BUFFEROVF;
1936 2585 2     : CH$COPY (.PTR [ITEM_LEN],
1937 2586 2     :     .DEV_DSC [DSC$A_POINTER],

```

```

1938 2587 4
1939 2588 4
1940 2589 4
1941 2590 4
1942 2591 4
1943 2592 4
1944 2593 4
1945 2594 4
1946 2595 4
1947 2596 4
1948 2597 4
1949 2598 4
1950 2599 4
1951 2600 4
1952 2601 4
1953 2602 4
1954 2603 4
1955 2604 4
1956 2605 4
1957 2606 4
1958 2607 4
1959 2608 4
1960 2609 4
1961 2610 5
1962 2611 5
1963 2612 5
1964 2613 5
1965 2614 5
1966 2615 5
1967 2616 5
1968 2617 5
1969 2618 5
1970 2619 4
1971 2620 4
1972 2621 3
1973 2622 3
1974 2623 3
1975 2624 4
1976 2625 4
1977 2626 4
1978 2627 4
1979 2628 4
1980 2629 4
1981 2630 4
1982 2631 4
1983 2632 4
1984 2633 4
1985 2634 4
1986 2635 4
1987 2636 4
1988 2637 4
1989 2638 4
1990 2639 4
1991 2640 4
1992 2641 4
1993 2642 4
1994 2643 4

```

[MNT\$_FLAGS] :

```

0,
.PTR [ITEM_LENGTH],
.STR_PTR);
DEVICE_COUNT = .DEVICE_COUNT + 1;
END;

BEGIN
: For FLAGS:
    a. Create item descriptor, relocate address
    b. Compute length, return SS$_BUFFEROVF if appropriate
    c. Copy flags, clear MNT$V_CLUSTER bit, and
       set MNT$V_NOASSIST (disables operator assist)
PTR [ITEM_LENGTH] = .ITEM [ITEM_LENGTH];
PTR [ITEM_CODE] = MNT$_FLAGS;
PTR [ITEM_ADDR] = .STR_PTR - .BUFFER;
.LENGTH = .LENGTH + ITEM_SIZE + .PTR [ITEM_LENGTH];
IF .LENGTH GTRU .LIMIT
THEN
    RETURN SS$_BUFFEROVF;
BEGIN
    BIND
        TEMP_PTR = .STR_PTR : BBLOCK;
        TEMP_PTR = .ITEM [ITEM_ADDR];
        TEMP_PTR [MNT$V_CLUSTER] = 0;
        TEMP_PTR [MNT$V_NOASSIST] = 1;
        IF NOT .MOUNT_OPTIONS [OPT_GROUP]
        THEN
            TEMP_PTR [MNT$V_SYSTEM] = 1;
END;
END;

```

[OTHERWISE] :

```

BEGIN
: All others:
    a. Create item descriptor, relocate address
    b. Compute length, return SS$_BUFFEROVF if appropriate
    c. Copy item to output buffer
PTR [ITEM_LENGTH] = .ITEM [ITEM_LENGTH];
PTR [ITEM_CODE] = .ITEM [ITEM_CODE];
PTR [ITEM_ADDR] = .STR_PTR - .BUFFER;
.LENGTH = .LENGTH + ITEM_SIZE + .PTR [ITEM_LENGTH];
IF .LENGTH GTRU .LIMIT
THEN
    RETURN SS$_BUFFEROVF;
CH$COPY (.ITEM [ITEM_LENGTH],
        .ITEM [ITEM_ADDR],
        0,
        .ITEM [ITEM_LENGTH],
        .STR_PTR);

```

```

: 1995      2644      3      END.
: 1996      2645
: 1997      2646
: 1998      2647
: 1999      2648
: 2000      2649      TES;
: 2001      2650      STR_PTR = .STR_PTR + .PTR [ITEM_LEN];      ! Bump string buffer pointer
: 2002      2651      ITEM = .ITEM + ITEM_SIZE;      ! Bump item descriptor pointer
: 2003      2652      PTR = .PTR + ITEM_SIZE;      ! Bump output descriptor pointer
: 2004      2653      END;      ! End of item list loop
: 2005      2654
: 2006      2655      RETURN 1;
: 2007      2656      1 END;      ! End of MOUNT_ENCIPHER

```

```

OFFC 0000 MOUNT_ENCIPHER:
SE      08      C2      00002      .WORD      Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11      : 2417
59      04      AC      D0      00005      SUBL2      #8, SP
59      50      D4      00009      MOVL      ITEM_LIST, ITEM      : 2538
02      A9      B5      0000B      1$:      CLRL      ITEM_COUNT      : 2539
07      13      0000E      TSTW      2(ITEM)      : 2540
50      D6      00010      BEQL      2$
59      0C      C0      00012      INCL      ITEM_COUNT      : 2542
56      50      0C      C5      00017      2$:      ADDL2      #12, -ITEM      : 2543
5B      56      0C      AC      C1      0001B      BRB      1$      : 2540
58      0C      AC      D0      00022      MULL3      #12, ITEM_COUNT, R6      : 2549
59      04      AC      D0      00026      ADDL3      BUFFER, R6, STR_PTR      : 2550
5A      10      AC      D0      0002C      CLRL      (STR_PTR)+      : 2552
6A      04      04      D0      00030      MOVL      BUFFER, PTR      : 2553
56      01      A0      9E      00033      MOVL      ITEM_LIST, ITEM      : 2554
00B3    31      00037      CLRL      DEVICE_COUNT      : 2555
6E      02      A9      3C      0003A      3$:      MOVL      LENGTH, R10      : 2561
04      01      01      D0      0003E      MOVAB     1(R0), J      : 2563
6E      6E      B1      00042      BRW      9$
01      33      12      00045      MOVZWL   2(ITEM), (SP)      : 2566
04      04      AE      D4      00047      MOVL     #1, 4(SP)
57      01      78      0004A      CMPW     (SP), #1      : 2576
50      50      00000000'EF40 DE 0004E      BNEQ     4$
68      60      B0      00056      CLRL     4(SP)
04      A8      01      B0      00059      ASHL     #1, DEVICE_COUNT, R0      : 2578
5B      0C      AC      C3      0005D      MOVAL   PHYS_NAME[R0], R0      : 2579
51      68      3C      00063      MOVW     (R0), (PTR)
51      6A      C0      00066      MOVW     #1, 2(PTR)      : 2580
6A      0C      A1      9E      00069      SUBL3   BUFFER, STR_PTR, 4(PTR)      : 2581
08      AC      6A      D1      0006D      MOVZWL  (PTR), R1
68      04      B0      28      00073      ADDL2   (R10), R1
04      6E      B1      0007A      4$:      MOVAB   12(R1), (R10)
63      1A      00071      CML     (R10), LIMIT      : 2582
68      28      00073      BGTRU   6$
57      D6      00078      MOVC3   (PTR), 24(R0), (STR_PTR)      : 2589
04      6E      B1      0007A      INCL    DEVICE_COUNT      : 2590
36      12      0007D      CMPW    (SP), #4      : 2593

```

| | | | | | | | | | | | |
|----|----|----|----|-------|------|-------|-------|--------|-----------------------------|--|------|
| | | | 04 | AE | D4 | 0007F | | CLRL | 4(SP) | | |
| | | 68 | | 69 | B0 | 00082 | | MOVW | (ITEM), (PTR) | | 2603 |
| | | A8 | 02 | 04 | B0 | 00085 | | MOVW | #4, 2(PTR) | | 2604 |
| 04 | A8 | 5B | | 0C | AC | 00089 | | SUBL3 | BUFFER, STR_PTR, 4(PTR) | | 2605 |
| | | 50 | | | 68 | 0008F | | MOVZWL | (PTR), R0 | | 2606 |
| | | 50 | | | 6A | 00092 | | ADDL2 | (R10), R0 | | |
| | | 6A | 08 | 0C | A0 | 00095 | | MOVAB | 12(R0), (R10) | | |
| | | AC | | | 6A | 00099 | | CMPL | (R10), LIMIT | | 2607 |
| | | | | | 37 | 0009D | | BGTRU | 6\$ | | |
| | | 6B | | 04 | B9 | 0009F | | MOVL | (ITEM), (STR_PTR) | | 2613 |
| | | AB | 03 | | 10 | 000A3 | | BICB2 | #16, 3(STR_PTR) | | 2614 |
| | | 6B | | | 04 | 000A7 | | BISB2 | #4, (STR_PTR) | | 2615 |
| | | | | 0000G | CF | 000AA | | TSTB | MOUNT_OPTIONS | | 2616 |
| | | | | | 05 | 000AE | | BLSS | 5\$ | | |
| | | AB | 01 | 40 | 8F | 000B0 | | BISB2 | #64, 1(STR_PTR) | | 2618 |
| | | 28 | | 04 | AE | 000B5 | 5\$: | BLBC | 4(SP), 8\$ | | 2623 |
| | | 6B | | | 69 | 000B9 | | MOVW | (ITEM), (PTR) | | 2632 |
| | | A8 | 02 | | 6E | 000BC | | MOVW | (SP), 2(PTR) | | 2633 |
| 04 | A8 | 5B | | 0C | AC | 000C0 | | SUBL3 | BUFFER, STR_PTR, 4(PTR) | | 2634 |
| | | 50 | | | 68 | 000C6 | | MOVZWL | (PTR), R0 | | 2635 |
| | | 50 | | | 6A | 000C9 | | ADDL2 | (R10), R0 | | |
| | | 6A | 08 | 0C | A0 | 000CC | | MOVAB | 12(R0), (R10) | | |
| | | AC | | | 6A | 000D0 | | CMPL | (R10), LIMIT | | 2636 |
| | | | | | 06 | 000D4 | | BLEQU | 7\$ | | |
| | | 50 | | 0601 | 8F | 000D6 | 6\$: | MOVZWL | #1537, R0 | | 2638 |
| | | | | | 04 | 000DB | | RET | | | |
| | 6B | | 04 | | 69 | 000DC | 7\$: | MOV3 | (ITEM), @4(ITEM), (STR_PTR) | | 2643 |
| | | 50 | | | 88 | 000E1 | 8\$: | MOVZWL | (PTR)+, R0 | | 2648 |
| | | 5B | | | 50 | 000E4 | | ADDL2 | R0, STR_PTR | | |
| | | 59 | | | 0C | 000E7 | | ADDL2 | #12, ITEM | | 2649 |
| | | 58 | | | 0A | 000EA | | ADDL2 | #10, PTR | | 2650 |
| | | 02 | | | 56 | 000ED | 9\$: | SOBGTR | J, 10\$ | | 2561 |
| | | | | | 03 | 000F0 | | BRB | 11\$ | | |
| | | | | | FF45 | 000F2 | 10\$: | BRW | 3\$ | | |
| | | 50 | | | 01 | 000F5 | 11\$: | MOVL | #1, R0 | | 2655 |
| | | | | | 04 | 000F8 | | RET | | | 2656 |

; Routine Size: 249 bytes, Routine Base: \$CODE\$ + 0849

; 2008 2657 1

```

2010 2658 1
2011 2659 1 ROUTINE SEARCH_DEVICE (J) =
2012 2660 1
2013 2661 1 ++
2014 2662 1
2015 2663 1 FUNCTIONAL DESCRIPTION:
2016 2664 1
2017 2665 1 This routine searches the I/O database for a mountable device
2018 2666 1 and allocates the device, if it is not already allocated. If
2019 2667 1 the mount operation is a private mount, an EX mode lock will
2020 2668 1 be taken out. If the mount operation is a shared mount, a PW
2021 2669 1 mode lock will be taken out.
2022 2670 1
2023 2671 1 Note: this routine must be called in kernel mode.
2024 2672 1
2025 2673 1 CALLING SEQUENCE:
2026 2674 1
2027 2675 1 SEARCH_DEVICE (ARG1)
2028 2676 1
2029 2677 1 INPUT:
2030 2678 1
2031 2679 1 ARG1 : Index into device list.
2032 2680 1
2033 2681 1 OUTPUT:
2034 2682 1
2035 2683 1 None.
2036 2684 1
2037 2685 1 IMPLICIT INPUT:
2038 2686 1
2039 2687 1 Mount database.
2040 2688 1
2041 2689 1 IMPLICIT OUTPUT:
2042 2690 1
2043 2691 1 The physical device name of the device will be set up in
2044 2692 1 the mount data base, with the appropriate device descriptor
2045 2693 1 set up in PHYS_NAME.
2046 2694 1
2047 2695 1 ROUTINE VALUE:
2048 2696 1
2049 2697 1 Assorted status codes.
2050 2698 1
2051 2699 1 SIDE EFFECTS:
2052 2700 1
2053 2701 1 None.
2054 2702 1
2055 2703 1 NOTES:
2056 2704 1
2057 2705 1 To properly interlock the device in a cluster environment,
2058 2706 1 we must carefully take out the MOUS interlock and the device
2059 2707 1 lock to serialize the mounts in the cluster without deadlocks.
2060 2708 1 Following is the algorithm used:
2061 2709 1
2062 2710 1 Step 0: Lock I/O database;
2063 2711 1 IOC$SEARCH (...);
2064 2712 1 If success
2065 2713 1 then S0
2066 2714 1 else F0

```

```

: 2067      2715  1
: 2068      2716  1
: 2069      2717  1
: 2070      2718  1
: 2071      2719  1
: 2072      2720  1
: 2073      2721  1
: 2074      2722  1
: 2075      2723  1
: 2076      2724  1
: 2077      2725  1
: 2078      2726  1
: 2079      2727  1
: 2080      2728  1
: 2081      2729  1
: 2082      2730  1
: 2083      2731  1
: 2084      2732  1
: 2085      2733  1
: 2086      2734  1
: 2087      2735  1
: 2088      2736  1
: 2089      2737  1
: 2090      2738  1
: 2091      2739  1
: 2092      2740  1
: 2093      2741  1
: 2094      2742  1
: 2095      2743  1
: 2096      2744  1
: 2097      2745  1
: 2098      2746  1
: 2099      2747  1
: 2100      2748  1
: 2101      2749  1
: 2102      2750  1
: 2103      2751  1
: 2104      2752  1
: 2105      2753  1
: 2106      2754  1
: 2107      2755  1
: 2108      2756  1
: 2109      2757  1
: 2110      2758  1
: 2111      2759  1
: 2112      2760  1
: 2113      2761  1
: 2114      2762  1
: 2115      2763  1
: 2116      2764  1
: 2117      2765  1
: 2118      2766  1
: 2119      2767  1
: 2120      2768  1
: 2121      2769  1
: 2122      2770  1
: 2123      2771  1

```

```

S0:  If device allocated
      then
          set SSS_DEVALRALLOC
      else
          mark UCB as allocated;

S1:  IOC$CVT_DEVNAM (...) to convert device name;

S2:  Unlock I/O database;

S3:  $GETDVIW to obtain allocation class name;

S4:  $ENQW MOUS lock with LCK$M_NOQUEUE

S5:  If success
      then
          Exit loop;

S6:  If SSS_DEVALRALLOC
      then
          IOC$UNLOCK_DEV to dequeue device lock
      else
          IOC$DALLOC_DEV to deallocate and release device lock;

S7:  Wait delta time;

S8:  $ENQW MOUS lock;

S9:  $DEQ MOUS lock;

S10: Goto step 0;

F0:  If ( not SSS_DEVALLOC )
      or ( private_mount )
      or ( device_allocated )
      then
          Unlock I/O database;
          Exit loop;

F1:  IOC$CVT_DEVNAM (...) to convert alloc class device name;

F2:  Unlock I/O database;

F3:  Wait delta time;

F4:  $ENQW MOUS lock;

F5:  Construct device lock;
      $ENQW device in CR mode with NOQUEUE;

F6:  If failed
      then
          Exit loop;

F7:  $DEQ device lock;

```



```

2124 2772 1
2125 2773 1
2126 2774 1
2127 2775 1
2128 2776 1
2129 2777 1
2130 2778 1
2131 2779 1
2132 2780 1
2133 2781 1
2134 2782 1
2135 2783 1
2136 2784 1
2137 2785 1
2138 2786 1
2139 2787 2
2140 2788 2
2141 2789 2
2142 2790 2
2143 2791 2
2144 2792 2
2145 2793 2
2146 2794 2
2147 2795 2
2148 2796 2
2149 2797 2
2150 2798 2
2151 2799 2
2152 2800 2
2153 2801 2
2154 2802 2
2155 2803 2
2156 2804 2
2157 2805 2
2158 2806 2
2159 2807 2
2160 2808 2
2161 2809 2
2162 2810 2
2163 2811 2
2164 2812 2
2165 2813 2
2166 2814 2
2167 2815 2
2168 2816 2
2169 2817 2
2170 2818 2
2171 2819 2
2172 2820 2
2173 2821 2
2174 2822 2
2175 2823 2
2176 2824 2
2177 2825 2
2178 2826 2
2179 2827 2
2180 2828 2

```

```

F8:   $DEQ MOUS lock;
F9:   Goto step 0;

Note: This algorithm does not handle simultaneous /shared mounts
      from the same system. In this case, the first mouter will
      mount the device, and the second mouter will fail with an
      SSS_DEVALLOC error. To properly serialize simultaneous
      shared mounts, another level of lock (the label lock) had
      to be added.

--
BEGIN
LOCAL
SEARCH_FLAGS : BBLOCK [4],           ! IOC$SEARCH routine flags
UCB           : REF BBLOCK,         ! Address of the UCB
STATUS,       :                   ! Routine status
SEARCH_STATUS, :                   ! Final success status
COUNTER,      :                   ! Count number of iterations
DEVICE_ITMLST1 : BBLOCK [(1 * 12) + 4] INITIAL
               : item: allocation class plus device name
               : (WORD (NAMEBUF LEN-4),
               : WORD (DVI$ AL[DEVNAM]),
               : LONG (ALLDEVNAM_BUF+4),
               : LONG (ALLDEVNAM_DESC),
               : end of list
               : LONG (0));

EXTERNAL
DEV_CTX       : BBLOCK FIELD (DC),   ! device value block context fields
MOUNT_OPTIONS : BITVECTOR VOLATILE, ! Parser option flags
DEVICE_STRING : VECTOR VOLATILE,    ! Device name string descriptor
SCH$GL_CURPCB : REF BBLOCK ADDRESSING_MODE (GENERAL); ! PCB address of current process

LINKAGE
IOC_SEARCH = JSB (REGISTER = 1, REGISTER = 2, REG STER = 3,
                 REGISTER = 4; REGISTER = 1) :
                 NOPRESERVE (2, 3, 5),

IOC_CVT_DEVNAM = JSB (REGISTER = 0, REGISTER = 1, REGISTER = 4,
                    REGISTER = 5; REGISTER = 1) :
                    PRESERVE (2,3,4,5,6,7),

IOC_LOCK_DEV = JSB (REGISTER = 0, REGISTER = 1, REGISTER = 4,
                  REGISTER = 5) : NOPRESERVE (4, 5),

IOC_UNLOCK_DEV = JSB (REGISTER = 5),

IOC_DALLOC_DEV = JSB (REGISTER = 4, REGISTER = 5) :

```

```

2181 2829 2 NOPRESERVE (1,2,3,8),
2182 2830
2183 2831 EXE_MAXACMODE = JSB (REGISTER = 0) ;
2184 2832 NOTUSED (2,3,4,5,6,7,8,9,10,11);
2185 2833
2186 2834 EXTERNAL ROUTINE
2187 2835 LOCK_IODB : ADDRESSING_MODE (GENERAL),
2188 2836 ! Lock I/O database mutex
2189 2837 UNLOCK_IODB : ADDRESSING_MODE (GENERAL),
2190 2838 ! Unlock the above
2191 2839 IOCSSEARCH : IOC_SEARCH ADDRESSING_MODE (GENERAL),
2192 2840 ! Search I/O database for device
2193 2841 IOCSCVT_DEVNAM : IOC_CVT_DEVNAM ADDRESSING_MODE (GENERAL),
2194 2842 ! Get fully expanded device name
2195 2843 IOCSLOCK_DEV : IOC_LOCK_DEV ADDRESSING_MODE (GENERAL),
2196 2844 ! Take out the device lock
2197 2845 IOCSUNLOCK_DEV : IOC_UNLOCK_DEV ADDRESSING_MODE (GENERAL),
2198 2846 ! Release the device lock
2199 2847 IOCSDALLOC_DEV : IOC_DALLOC_DEV ADDRESSING_MODE (GENERAL),
2200 2848 ! Deallocate device and device lock
2201 2849 EXESMAXACMODE : EXE_MAXACMODE ADDRESSING_MODE (GENERAL);
2202 2850 ! Maximize access mode
2203 2851
2204 2852
2205 2853 ! Rebind things to make life easier ( so we see them as their
2206 2854 real logical units).
2207 2855
2208 2856 MAP
2209 2857 DEVICE_STRING : BBLOCKVECTOR [ DEVMAX, 8 ],
2210 2858 NAME_BUFFER : BBLOCKVECTOR [ DEVMAX, NAMEBUF_LEN ],
2211 2859 PHYS_NAME : BBLOCKVECTOR [ DEVMAX, 8 ];
2212 2860
2213 2861 ! Start of buffer
2214 2862
2215 2863 MACRO STADR = 0,0,0,0%;
2216 2864
2217 2865
2218 2866 ! Define descriptor vector displacements
2219 2867
2220 2868
2221 2869 MACRO LEN = 0,0,32,0%;
2222 2870 MACRO ADDR = 4,0,32,0%;
2223 2871 MACRO ILEN = 8,0,32,0%; ! Item list returned length position.
2224 2872
2225 2873 LITERAL
2226 2874 RETRY_LIMIT = 1000; ! Define retry limit
2227 2875
2228 2876 SEARCH_FLAGS [0,0,32,0] = 0; ! Initialize search flags
2229 2877 SEARCH_FLAGS [IOC$V_MOUNT] = 1; ! Set flag to indicate searching for a mountable device
2230 2878
2231 2879
2232 2880 ! If this is a private mount, set flag to take out an exclusive lock on
2233 2881 the device.
2234 2882
2235 2883 IF .MOUNT_OPTIONS [OPT_NOSHARE]
2236 2884 THEN
2237 2885 SEARCH_FLAGS [IOC$V_ALLOC] = 1;

```

```

2238 2886 2
2239 2887 2 COUNTER = 0; ! Initialize counter
2240 2888 2
2241 2889 2 WHILE (1) DO ! Forever do block
2242 2890 2
2243 2891 2 BEGIN
2244 2892 2
2245 2893 2 LOCK_IODB (); ! Lock I/O database
2246 2894 2 STATUS = IOC$SEARCH ( DEVICE_STRING [.J, LEN], ! Search device with proper flags
2247 2895 2 .SEARCH_FLAGS,
2248 2896 2 DEV_CTX; ! Return lock value block
2249 2897 2 .SCH$GL_CURPCB;
2250 2898 2 UCB ); ! Target device UCB
2251 2899 2
2252 2900 2 IF .STATUS
2253 2901 2 THEN
2254 2902 2
2255 2903 2 BEGIN ! IOC$SEARCH succeeded
2256 2904 2
2257 2905 2 ! If the device is not already allocated, allocate the device by
2258 2906 2 ! setting up the proper status in the I/O database.
2259 2907 2
2260 2908 2 IF NOT .BBLOCK [UCB [UCB$DEVCHAR], DEV$V_ALL]
2261 2909 2 THEN
2262 2910 2 BEGIN
2263 2911 2 UCB [UCB$B_AMOD] = EXE$MAXACMODE (.CALLERS_ACMOD); ! Set access mode
2264 2912 2 BBLOCK [UCB [UCB$DEVCHAR], DEV$V_ALL] = T; ! Set the device as allocated
2265 2913 2 UCB [UCB$W_REFC] = .UCB [UCB$W_REFC] + 1; ! Bump reference count
2266 2914 2 UCB [UCB$PID] = .SCH$GL_CURPCB [PCB$PID]; ! Set device owner
2267 2915 2 SEARCH_STATUS = SS$NORMAL; ! Set normal return status
2268 2916 2 END
2269 2917 2 ELSE
2270 2918 2 SEARCH_STATUS = SS$DEVALRALLOC; ! Set proper return status
2271 2919 2
2272 2920 2
2273 2921 2 ! Set up physical device name in mount database (also set up the device
2274 2922 2 ! descriptor).
2275 2923 2
2276 2924 2 IOC$CVT_DEVNAM ( NAMEBUF_LEN, ! Output buffer length
2277 2925 2 NAME_BUFFER [.J, STADR], ! Output buffer address
2278 2926 2 -1, ! Format device name
2279 2927 2 .UCB; ! Address of UCB
2280 2928 2 PHYS_NAME [.J, LEN] ); ! Returned length of device name
2281 2929 2
2282 2930 2 PHYS_NAME [.J, ADDR] = NAME_BUFFER [.J, STADR]; ! Set up device descriptor
2283 2931 2
2284 2932 2 UNLOCK_IODB (); ! Unlock I/O database
2285 2933 2
2286 2934 2 P $GETDVIW ( DEVNAM = PHYS_NAME [.J, LEN], ! Target device descriptor
2287 2935 2 P ITMLST = DEVICE_ITMLST1, ! Item list
2288 2936 2 EFN = MOUNT_EFN );
2289 2937 2 ALLDEVNAM_DESC [0] = .ALLDEVNAM_DESC [0] + 4; ! Fix up length to include MOUS
2290 2938 2
2291 2939 2
2292 2940 2 ! Take out a lock on the allocation class device name. This will
2293 2941 2 ! interlock all mounts of this device.
2294 2942 2

```

```

: 2295 P 2943 4 STATUS = $ENQW (LKMODE = LCK$K_EXMODE,
: 2296 P 2944 4 LKSB = LOCK_STATUS,
: 2297 P 2945 4 FLAGS = (LCK$M_SYSTEM OR LCK$M_NOQUEUE),
: 2298 P 2946 4 RESNAM = ALLDEVNAM_DESC,
: 2299 P 2947 4 EFN = MOUNT_EFN,
: 2300 2948 4 ACMODE = PSL$C_EXEC);
: 2301 2949 4 IF .STATUS
: 2302 2950 4 THEN
: 2303 2951 5 BEGIN ! MOUS interlock granted
: 2304 2952 5 STATUS = .SEARCH_STATUS; ! Return proper status code
: 2305 2953 5 EXITLOOP; ! Get out of the loop
: 2306 2954 5 END ! End of MOUS success story
: 2307 2955 4 ELSE
: 2308 2956 5 BEGIN ! MOUS interlock failed
: 2309 2957 5 LOCK_IODB (); ! Lock I/O database
: 2310 2958 5 IF .SEARCH_STATUS EQL S$$_DEVALRALLOC
: 2311 2959 5 THEN
: 2312 2960 5 IOCSUNLOCK_DEV (.UCB) ! Release device lock
: 2313 2961 5 ELSE
: 2314 2962 5 IOCS$DALLOC_DEV (.SCH$GL_CURPCB, .UCB); ! Deallocate device and
: 2315 2963 5 ! release device lock
: 2316 2964 5 UNLOCK_IODB (); ! Unlock I/O database
: 2317 2965 5
: 2318 2966 5 WAIT_DELTA (.COUNTER); ! Wait a while
: 2319 2967 5
: 2320 P 2968 5 $ENQW (LKMODE = LCK$K_EXMODE, ! Enqueue MOUS lock again
: 2321 P 2969 5 LKSB = LOCK_STATUS,
: 2322 P 2970 5 FLAGS = LCK$M_SYSTEM,
: 2323 P 2971 5 RESNAM = ALLDEVNAM_DESC,
: 2324 P 2972 5 EFN = MOUNT_EFN,
: 2325 2973 5 ACMODE = PSL$C_EXEC);
: 2326 2974 5 $DEQ ( LKID = .LOCK_STATUS [1] ); ! Dequeue MOUS lock
: 2327 2975 5
: 2328 2976 5 END ! End of MOUS failure block
: 2329 2977 5
: 2330 2978 4 END ! End of IOCS$SEACH success block
: 2331 2979 4
: 2332 2980 3 ELSE
: 2333 2981 3 BEGIN ! IOCS$SEARCH failure block
: 2334 2982 4
: 2335 2983 4
: 2336 2984 5 IF ( .STATUS NEQ S$$_DEVALLOC ) ! If not S$$_DEVALLOC
: 2337 2985 5 OR ( .MOUNT_OPTIONS [OPT_NOSHARE] ) ! or this is a private mount
: 2338 2986 5 OR ( .BBLOCK [ UCB [UCB$_DEVCHAR], DEV$V_ALL ] ) ! or an allocated device
: 2339 2987 4 THEN ! Get out
: 2340 2988 5 BEGIN
: 2341 2989 5 UNLOCK_IODB ();
: 2342 2990 5 EXITLOOP;
: 2343 2991 4 END;
: 2344 2992 4
: 2345 2993 4
: 2346 2994 4 ! We have a valid UCB address, get the allocation device name to
: 2347 2995 4 ! derive the MOUS interlock.
: 2348 2996 4
: 2349 2997 4 IOCS$CVT_DEVNAM ( NAMEBUF_LEN-4, ! Output buffer length
: 2350 2998 4 ALLDEVNAM_BUF+4, ! Output buffer address
: 2351 2999 4 1, ! Format allocation class device name

```

```

2352 3000 4
2353 3001 4
2354 3002 4
2355 3003 4
2356 3004 4
2357 3005 4
2358 3006 4
2359 3007 4
2360 3008 4
2361 3009 4
2362 3010 4
2363 3011 4
2364 3012 4
2365 3013 4
2366 3014 4
2367 3015 4
2368 3016 4
2369 3017 4
2370 3018 4
2371 3019 4
2372 3020 4
2373 3021 4
2374 3022 5
2375 3023 5
2376 3024 5
2377 3025 5
2378 3026 5
2379 3027 5
2380 3028 5
2381 3029 5
2382 3030 5
2383 3031 5
2384 3032 5
2385 3033 5
2386 3034 5
2387 3035 5
2388 3036 5
2389 3037 5
2390 3038 5
2391 3039 5
2392 3040 5
2393 3041 5
2394 3042 5
2395 3043 5
2396 3044 5
2397 3045 5
2398 3046 5
2399 3047 5
2400 3048 5
2401 3049 5
2402 3050 5
2403 3051 6
2404 3052 6
2405 3053 6
2406 3054 6
2407 3055 5
2408 3056 5

```

P
P
P
P
P

P
P
P
P

```

      .URB;
      ALLDEVNAM_DESC [0]);
      ! Address of UCB
      ! Returned length of device name
ALLDEVNAM_DESC [0] = .ALLDEVNAM_DESC [0] + 4; ! Fix up length to include MOUS

UNLOCK_IODB ();
      ! Unlock I/O database

WAIT_DELTA (.COUNTER);
      ! Wait a while

      ! Take out a lock on the allocation class device name.
$ENQW (LKMODE = LCK$K_EXMODE,
      LKSB = LOCK_STATUS,
      FLAGS = LCK$M_SYSTEM,
      RESNAM = ALLDEVNAM_DESC,
      EFN = MOUNT_EFN,
      ACMODE = PSL$C_EXEC);

      ! Construct the device lock name and take out the device lock
      ! in CR mode with NOQUEUE.
BEGIN
LOCAL
  DEVLCKNAM_BUF : VECTOR [NAMEBUF_LEN, BYTE]
                  INITIAL (BYTE('SYSS', REP NAMEBUF_LEN-4 OF (' '))),
  DEVLCKNAM_DESC : VECTOR [2, LONG]
                  INITIAL (0, DEVLCKNAM_BUF),
  DEVLCK_STS : VECTOR [2, LONG];

DEVLCKNAM_DESC [0] = .ALLDEVNAM_DESC [0]; ! Set up device lock descriptor
CH$COPY ( .ALLDEVNAM_DESC [0] - 4, ! Length of input string
          .ALLDEVNAM_DESC [1] + 4, ! Start of alloc name string
          0,
          .ALLDEVNAM_DESC [0] - 4, ! Length of output string
          .DEVLCKNAM_DESC [1] + 4 ); ! Start of target string

STATUS = $ENQW (LKMODE = LCK$K_CRMODE, ! Enqueue device lock in CR mode
               LKSB = DEVLCK_STS, ! Lock status block
               FLAGS = (LCK$M_SYSTEM OR LCK$M_NOQUEUE), ! Return if not available
               RESNAM = DEVLCKNAM_DESC, ! Device lock
               EFN = MOUNT_EFN);

IF .STATUS
THEN
      ! Device lock in CR mode granted. This implies that the device
      ! is not allocated. Release both locks and try again.
      BEGIN
      $DEQ ( LKID = .DEVLCK_STS [1] ); ! Release device lock
      $DEQ ( LKID = .LOCK_STATUS [1] ); ! Release MOUS lock
      END
ELSE
      !

```

```

2409 3057 5      : Device lock in CR mode is not granted. This will happen if the
2410 3058 5      : lock is already taken out in EX mode, i.e. the device is allocated.
2411 3059 5      : Get out with an SSS_DEVALLOC status.
2412 3060 5
2413 3061 6
2414 3062 6      BEGIN
2415 3063 6      STATUS = SSS_DEVALLOC;           ! Set return code
2416 3064 5      EXITLOOP;                   ! Get out
2417 3065 5      END;
2418 3066 4      END;                       ! End of block defining DEVLCK
2419 3067 4
2420 3068 3      END;                       ! End of IOC$SEARCH failure block
2421 3069 3
2422 3070 3
2423 3071 3      : Do a sanity check on how many times we have gone thru this loop. If
2424 3072 3      : too many times, give up with an error.
2425 3073 3
2426 3074 3      COUNTER = .COUNTER + 1.       ! Update counter
2427 3075 3      IF .COUNTER GEQ RETRY_LIMIT   ! If loop thru too many times
2428 3076 3      THEN                          ! give up with an error
2429 3077 4          BEGIN
2430 3078 4          STATUS = SSS_DEVNOTMOUNT;
2431 3079 4          EXITLOOP;
2432 3080 4          END;
2433 3081 3
2434 3082 2      END;                       ! End of forever block
2435 3083 2
2436 3084 2
2437 3085 2      IF NOT .STATUS
2438 3086 2      THEN                          ! If SEARCH_DEVICE failed
2439 3087 2          BEGIN
2440 3088 2
2441 3089 2          LOCAL
2442 3090 2              ITMLST2                : BBLOCK [(1 * 12) + 4] INITIAL
2443 3091 2              :
2444 3092 2              : item: device name
2445 3093 2              :
2446 3094 2              (WORD (NAMEBUF LEN),   ! Device name buffer length
2447 3095 2              WORD (DVIS_DEVNAM),    ! Device name item code
2448 3096 2              LONG (0),              ! Device name buffer address
2449 3097 2              LONG (0),              ! Returned device name length
2450 3098 2              :
2451 3099 2              : end of list
2452 3100 2              :
2453 3101 2              LONG (0)),
2454 3102 2          LOC_STATUS;                ! Local status work
2455 3103 2
2456 3104 2
2457 3105 2      : The IOC$SEARCH routine failed, use input device string to get the
2458 3106 2      : device name. Also set up the device descriptor. This is necessary
2459 3107 2      : so Operator Assist can output the message with a device name. If
2460 3108 2      : the $GETDVI failed, we've got some real problems, return the status
2461 3109 2      : as the status of routine SEARCH_DEVICE.
2462 3110 2
2463 3111 2      ITMLST2 [ADDR] = NAME_BUFFER [.J, STADR]; ! Set up device buffer address
2464 3112 2      ITMLST2 [ILEN] = PHYS_NAME [.J, LEN];    ! Set returned length
2465 3113 2      PHYS_NAME [.J, ADDR] = NAME_BUFFER [.J, STADR]; ! Set up descriptor

```

```
: 2466      3114  
: 2467      3115  
: 2468      3116  
: 2469      3117  
: 2470      3118  
: 2471      3119  
: 2472      3120  
: 2473      3121  
: 2474      3122  
: 2475      3123  
: 2476      3124  
: 2477      3125  
: 2478      3126
```

```
LOC_STATUS = $GETDVIW (DEVNAM = DEVICE_STRING [.J, LEN], ! Target device descriptor  
                     ITMLST = ITMLST2, ! Item list  
                     EFN = MOUNT_EFN );  
  
IF NOT .LOC_STATUS ! If we can't even get the device name  
THEN ! Return the status from $GETDVI  
    STATUS = .LOC_STATUS  
  
END; ! End of SEARCH_DEVICE failure block  
  
RETURN .STATUS; ! Return status  
END; ! End of routine SEARCH_DEVICE
```

```
.PSECT $PLITS,NOWRT,NOEXE,2  
  
001C 00020 P.AAC: .WORD 28  
00EC 00022 .WORD 236  
00000000 00024 .ADDRESS ALLDEVNAM_BUF+4  
00000000 00028 .ADDRESS ALLDEVNAM_DESC  
00000000 0002C  
24 53 59 53 00030 P.AAD: .ASCII \SYSS\  
20 00034 .ASCII //  
20 00035 .ASCII //  
20 00036 .ASCII //  
20 00037 .ASCII //  
20 00038 .ASCII //  
20 00039 .ASCII //  
20 0003A .ASCII //  
20 0003B .ASCII //  
20 0003C .ASCII //  
20 0003D .ASCII //  
20 0003E .ASCII //  
20 0003F .ASCII //  
20 00040 .ASCII //  
20 00041 .ASCII //  
20 00042 .ASCII //  
20 00043 .ASCII //  
20 00044 .ASCII //  
20 00045 .ASCII //  
20 00046 .ASCII //  
20 00047 .ASCII //  
20 00048 .ASCII //  
20 00049 .ASCII //  
20 0004A .ASCII //  
20 0004B .ASCII //  
20 0004C .ASCII //  
20 0004D .ASCII //  
20 0004E .ASCII //  
20 0004F .ASCII //  
0020 00050 P.AAE: .WORD 32  
0020 00052 .WORD 32  
00000000 00054 .LONG 0  
00000000 00058 .LONG 0  
00000000 0005C .LONG 0
```

| | | | | | | |
|----|-----------|-----------|----------------------------|--------|--------------------------------------|--------|
| | | | | .EXTRN | LOCK IODB, UNLOCK IODB | |
| | | | | .EXTRN | IOC\$SEARCH, IOC\$CVT DEVNAM | |
| | | | | .EXTRN | IOC\$LOCK DEV, IOC\$UNLOCK DEV | |
| | | | | .EXTRN | IOC\$DALLOC DEV, EXE\$MAXACMODE | |
| | | | | .EXTRN | SYSSGETDVIQ | |
| | | | | .PSECT | \$CODE\$,NOWRT,2 | |
| | | | | OFFC | 00000 SEARCH_DEVICE: | |
| | | | | .WORD | Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11 | : 2659 |
| 38 | AE | 00000000' | 5E 88 AE 9E 00002 | MOVAB | -72(SP), SP | |
| | | | EF 10 28 00006 | MOVCS | #16, P.AAC, DEVICE_ITMLST1 | : 2806 |
| | | | | CLRL | SEARCH_FLAGS | : 2876 |
| | 04 | 0000G | 6E 80 8F 88 00011 | BISB2 | #128, SEARCH_FLAGS | : 2877 |
| | | 01 | CF 04 E1 00015 | BBC | #4, MOUNT_OPTIONS, 1\$ | : 2883 |
| | | | AE 04 88 0001B | BISB2 | #4, SEARCH_FLAGS+1 | : 2885 |
| | | | | CLRL | COUNTER | : 2887 |
| | | 04 | 57 04 AC D0 00021 | MOVL | J, R7 | : 2894 |
| | | 00000000G | AE 0000GCF47 7E 00025 | MOVAB | DEVICE_STRING[R7], 4(SP) | |
| | | | 00 00 FB 0002C | CALLS | #0, LOCK_IODB | : 2893 |
| | | | 53 0000G CF 9E 00033 | MOVAB | DEV_CTX, R3 | : 2894 |
| | | | 54 00000000G 00 D0 00038 | MOVL | SCH\$GL_CURPCB, R4 | |
| | | | 52 6E D0 0003F | MOVL | SEARCH_FLAGS, R2 | |
| | | | 51 04 AE D0 00042 | MOVL | 4(SP), R1 | |
| | | | 00000000G 00 16 00046 | JSB | IOC\$SEARCH | |
| | | | 5A 50 D0 0004C | MOVL | R0, STATUS | |
| | | | 56 51 D0 0004F | MOVL | R1, R6 | |
| | | | 03 5A E8 00052 | BLBS | STATUS, 3\$ | : 2900 |
| | | | | BRW | 9\$ | |
| | | | 0117 31 00055 | TSTB | 58(UCB) | : 2908 |
| | | | 3A A6 95 00058 | BLSS | 4\$ | |
| | | | 2A 19 0005B | MOVL | CALLERS_ACMOD, R0 | : 2911 |
| | | | 50 00000000' EF D0 0005D | JSB | EXE\$MAXACMODE | |
| | | | 00000000G 00 16 00064 | MOVAB | R0, 95(UCB) | |
| | SF | | A6 50 90 0006A | BISB2 | #128, 58(UCB) | : 2912 |
| | 3A | | A6 80 8F 88 0006E | INCW | 92(UCB) | : 2913 |
| | | | 5C A6 B6 00073 | MOVL | SCH\$GL_CURPCB, R0 | : 2914 |
| | | | 50 00000000G 00 D0 00076 | MOVL | 96(R0), 44(UCB) | |
| | 2C | | A6 60 A0 D0 0007D | MOVL | #1, SEARCH_STATUS | : 2915 |
| | | | 5B 01 D0 00082 | BRB | 5\$ | : 2908 |
| | | | | BRB | 5\$ | : 2908 |
| | | | 58 0641 8F 3C 00087 | MOVZWL | #1601, SEARCH_STATUS | : 2918 |
| 52 | | | 57 05 78 0008C | ASHL | #5, R7, R2 | : 2925 |
| | | | 51 00000000' EF42 9E 00090 | MOVAB | NAME_BUFFER[R2], R1 | |
| | | | 55 56 D0 00098 | MOVL | UCB, R5 | : 2928 |
| | | | 54 01 CE 0009B | MNEGL | #1, R4 | |
| | | | 50 20 D0 0009E | MOVL | #32, R0 | |
| | | | 00000000G 00 16 000A1 | JSB | IOC\$CVT DEVNAM | |
| | | | 00000000' EF47 7F 000A7 | PUSHAQ | PHYS_NAME[R7] | |
| | | | 9E 51 D0 000AE | MOVL | R1, a(SP)+ | |
| | | | 00000000' EF47 7F 000B1 | PUSHAQ | PHYS_NAME+4[R7] | : 2930 |
| | | | 9E 00000000' EF42 9E 000B8 | MOVAB | NAME_BUFFER[R2], a(SP)+ | |
| | 00000000G | | 00 00 FB 000C0 | CALLS | #0, UNLOCK_IODB | : 2932 |
| | | | | CLRL | -(SP) | : 2936 |
| | | | | CLRL | -(SP) | |
| | | | 48 AE 9F 000CB | PUSHAB | DEVICE_ITMLST1 | |
| | | | 00000000' EF47 7F 000CE | PUSHAQ | PHYS_NAME[R7] | |

| | | | | | | | |
|-----------|-----------|-----------|----|-------|-------------|-----------------------|-------------------------|
| 00000000G | 7E | 1A | 7D | 000D5 | MOVQ | #26, -(SP) | |
| 00000000' | 00 | 08 | FB | 000D8 | CALLS | #8, SYSSGETDVIW | |
| | EF | 04 | CO | 000DF | ADDL2 | #4, ALLDEVNAM_DESC | 2937 |
| | 7E | 01 | 7D | 000E6 | MOVQ | #1, -(SP) | 2948 |
| | | 7E | 7C | 000E9 | CLRQ | -(SP) | |
| | | 7E | 7C | 000EB | CLRQ | -(SP) | |
| | 00000000' | EF | 9F | 000EC | PUSHAB | ALLDEVNAM_DESC | |
| | | 14 | DD | 000F3 | PUSHL | #20 | |
| | 00000000' | EF | 9F | 000F5 | PUSHAB | LOCK_STATUS | |
| | | 05 | DD | 000FB | PUSHL | #5 | |
| 00000000G | 00 | 1A | DD | 000FD | PUSHL | #26 | |
| | 5A | 08 | FB | 000FF | CALLS | #11, SYSSENQW | |
| | 05 | 50 | DO | 00106 | MCVL | R0, STATUS | |
| | 5A | 5A | E9 | 00109 | BLBC | STATUS, 6\$ | 2949 |
| | | 5B | DO | 0010C | MOVL | SEARCH_STATUS, STATUS | 2952 |
| 00000000G | 00 | 79 | 11 | 0010F | BRB | 11\$ | 2951 |
| 00000641 | 8F | 00 | FB | 00111 | 6\$: CALLS | #0, LOCK_IODB | 2957 |
| | | 5B | D1 | 00118 | CMPL | SEARCH_STATUS, #1601 | 2958 |
| | | 08 | 12 | 0011F | BNEQ | 7\$ | |
| | 55 | 56 | DO | 00121 | MOVL | UCB, R5 | 2960 |
| | 00000000G | 00 | 16 | 00124 | JSB | IOC\$UNLOCK_DEV | |
| | | 10 | 11 | 0012A | BRB | 8\$ | |
| | 55 | 56 | DO | 0012C | 7\$: MOVL | UCB, R5 | 2962 |
| | 54 | 00000000G | 00 | DO | 0012F | MOVL | SCH\$GL_CURPCB, R4 |
| | | 00000000G | 00 | 16 | 00136 | JSB | IOC\$DA[LOC_DEV |
| 00000000G | 00 | 00 | FB | 0013C | 8\$: CALLS | #0, UNLOCK_IODB | 2964 |
| | | 59 | DD | 00143 | PUSHL | COUNTER | 2966 |
| 00000000V | EF | 01 | FB | 00145 | CALLS | #1, WAIT_DELTA | |
| | 7E | 01 | 7D | 0014C | MOVQ | #1, -(SPT) | 2973 |
| | | 7E | 7C | 0014F | CLRQ | -(SP) | |
| | | 7E | 7C | 00151 | CLRQ | -(SP) | |
| | 00000000' | EF | 9F | 00153 | PUSHAB | ALLDEVNAM_DESC | |
| | | 10 | DD | 00159 | PUSHL | #16 | |
| | 00000000' | EF | 9F | 0015B | PUSHAB | LOCK_STATUS | |
| | | 05 | DD | 00161 | PUSHL | #5 | |
| | | 1A | DD | 00163 | PUSHL | #26 | |
| 00000000G | 00 | 08 | FB | 00165 | CALLS | #11, SYSSENQW | |
| | | 00D1 | 31 | 0016C | BRW | 13\$ | 2974 |
| 00000840 | 8F | 5A | D1 | 0016F | 9\$: CMPL | STATUS, #2112 | 2984 |
| | | 08 | 12 | 00176 | BNEQ | 10\$ | |
| 05 | 0000G | CF | 04 | E0 | 00178 | BBS | #4, MOUNT_OPTIONS, 10\$ |
| | | 3A | A6 | 95 | 0017E | TSTB | 58(UCB) |
| | | 0A | 18 | 00181 | BGEQ | 12\$ | 2986 |
| 00000000G | 00 | 00 | FB | 00183 | 10\$: CALLS | #0, UNLOCK_IODB | 2989 |
| | | 00DF | 31 | 0018A | 11\$: BRW | 17\$ | 2988 |
| | 51 | 00000000' | EF | 9E | 0018D | 12\$: MOVAB | ALLDEVNAM_BUF+4, R1 |
| | 55 | | 56 | DO | 00194 | MOVL | UCB, R5 |
| | 54 | | 01 | DO | 00197 | MOVL | #1, R4 |
| | 50 | | 1C | DO | 0019A | MOVL | #28, R0 |
| | 0000C000G | 00 | 16 | 0019D | JSB | IOC\$CVT_DEVNAM | |
| 00000000' | EF | 51 | DO | 001A3 | MOVL | R1, ALLDEVNAM_DESC | 3001 |
| 00000000' | EF | 04 | CO | 001AA | ADDL2 | #4, ALLDEVNAM_DESC | 3002 |
| 00000000G | 00 | 00 | FB | 001B1 | CALLS | #0, UNLOCK_IODB | 3004 |
| | | 59 | DD | 001B8 | PUSHL | COUNTER | 3006 |
| 00000000V | EF | 01 | FB | 001BA | CALLS | #1, WAIT_DELTA | |
| | 7E | 01 | 7D | 001C1 | MOVQ | #1, -(SPT) | 3016 |
| | | 7E | 7C | 001C4 | CLRQ | -(SP) | |

| | | | | | | | | | | |
|----|----|-----------|-----------|-----------|-----------|-----------|--------|---------------------------------|--|------|
| | | | 00000000' | 7E | 7C | 001C6 | CLRQ | -(SP) | | |
| | | | | EF | 9F | 001C8 | PUSHAB | ALLDEVNAM_DESC | | |
| | | | 00000000' | 10 | DD | 001CE | PUSHL | #16 | | |
| | | | | EF | 9F | 001D0 | PUSHAB | LOCK_STATUS | | |
| | | | | 05 | DD | 001D6 | PUSHL | #5 | | |
| | | | | 1A | DD | 001D8 | PUSHL | #26 | | |
| 18 | AE | 00000000G | 00 | 0B | FB | 001DA | CALLS | #11, SYS\$ENQW | | |
| | | | | 20 | 28 | 001E1 | MOV C3 | #32, P.AAD, DEVLCKNAM_BUF | | 3026 |
| | | | | AE | D4 | 001EA | CLRL | DEVLCKNAM_DESC | | |
| | | | 10 | AE | 9E | 001ED | MOVAB | DEVLCKNAM_BUF, DEVLCKNAM_DESC+4 | | |
| | | 14 | AE | 00000000' | EF | D0 | MOV L | ALLDEVNAM_DESC, DEVLCKNAM_DESC | | 3031 |
| | | 10 | AE | 00000000' | EF | D0 | MOV L | ALLDEVNAM_DESC, DEVLCKNAM_DESC | | 3033 |
| 52 | | 00000000' | | 04 | C3 | 001FA | SUBL3 | #4, ALLDEVNAM_DESC, R2 | | 3034 |
| | | | | 51 | 00000000' | EF | MOV L | ALLDEVNAM_DESC+4, R1 | | 3037 |
| | | | | 50 | | | MOV L | DEVLCKNAM_DESC+4, R0 | | |
| 04 | A0 | 04 | A1 | 14 | AE | D0 | MOV C3 | R2, 4(R1), 4(R0) | | |
| | | | | 52 | 28 | 0020D | CLRQ | -(SP) | | 3043 |
| | | | | 7E | 7C | 00213 | CLRQ | -(SP) | | |
| | | | | 7E | 7C | 00215 | CLRQ | -(SP) | | |
| | | | | 7E | 7C | 00217 | CLRQ | -(SP) | | |
| | | | | 28 | AE | 9F | PUSHAB | DEVLCKNAM_DESC | | |
| | | | | 14 | DD | 0021C | PUSHL | #20 | | |
| | | | | 28 | AE | 9F | PUSHAB | DEVLCK_STS | | |
| | | | | 01 | DD | 00221 | PUSHL | #1 | | |
| | | | | 1A | DD | 00223 | PUSHL | #26 | | |
| | | 00000000G | 00 | 0B | FB | 00225 | CALLS | #11, SYS\$ENQW | | |
| | | | 5A | 50 | D0 | 0022C | MOV L | R0, STATUS | | |
| | | | 21 | 5A | E9 | 0022F | BLBC | STATUS, 14\$ | | 3045 |
| | | | | 7E | 7C | 00232 | CLRQ | -(SP) | | 3052 |
| | | | | 7E | D4 | 00234 | CLRL | -(SP) | | |
| | | | | 18 | AE | DD | PUSHL | DEVLCK_STS+4 | | |
| | | 00000000G | 00 | 04 | FB | 00239 | CALLS | #4, SYS\$DEQ | | |
| | | | | 7E | 7C | 00240 | CLRQ | -(SP) | | 3053 |
| | | | | 7E | D4 | 00242 | CLRL | -(SP) | | |
| | | | | 00000000' | EF | DD | PUSHL | LOCK_STATUS+4 | | |
| | | | | 04 | FB | 0024A | CALLS | #4, SYS\$DEQ | | |
| | | | | 07 | 11 | 00251 | BRB | 15\$ | | 3045 |
| | | | | 5A | 0840 | 8F | MOVZWL | #2112, STATUS | | 3062 |
| | | | | 12 | 11 | 00258 | BRB | 17\$ | | 3061 |
| | | | | 59 | D6 | 0025A | INCL | COUNTER | | 3074 |
| | | 000003E8 | 8F | 59 | D1 | 0025C | CMP L | COUNTER, #1000 | | 3075 |
| | | | | 03 | 18 | 00263 | BGEQ | 16\$ | | |
| | | | | FDC4 | 31 | 00265 | BRW | 2\$ | | |
| | | | | 5A | 7C | 8F | MOVZBL | #124, STATUS | | 3078 |
| | | | | 46 | 5A | E8 | BLBS | STATUS, 18\$ | | 3085 |
| 28 | AE | 00000000' | | EF | 10 | 28 | MOV C3 | #16, P.AAE, ITMLST2 | | 3101 |
| | | | | 57 | 05 | 78 | ASHL | #5, R7, R2 | | 3111 |
| | | | | 50 | 00000000' | EF | MOVAB | NAME_BUFFER[R2], R0 | | |
| | | | | 2C | AE | 50 | MOV L | R0, ITMLST2+4 | | |
| | | | | 30 | AE | 00000000' | MOVAB | PHYS_NAME[R7], ITMLST2+8 | | 3112 |
| | | | | 00000000' | EF | 47 | PUSHAQ | PHYS_NAME+4[R7] | | 3113 |
| | | | | 9E | 50 | D0 | MOV L | R0, 3(SP)+ | | |
| | | | | | 7E | 7C | CLRQ | -(SP) | | 3117 |
| | | | | | 7E | 7C | CLRQ | -(SP) | | |
| | | | | 38 | AE | 9F | PUSHAB | ITMLST2 | | |
| | | | | 18 | AE | DD | PUSHL | 24(SP) | | |
| | | | | 7E | 1A | 7D | MOVQ | #26, -(SP) | | |
| | | 00000000G | 00 | 08 | FB | 002A8 | CALLS | #8, SYS\$GETDVIW | | |
| | | | 03 | 50 | E8 | 002AF | BLBS | LOC_STATUS, 18\$ | | 3119 |

VMOUNT
V04-002

K 9
16-Sep-1984 01:00:56
12-Sep-1984 11:14:53

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[MOUNT.SRC]VMOUNT.B32;3 (15) Page 73

| | | | | | | | |
|----|----|----|-------|------|--------------------|---|------|
| 5A | 50 | DO | 002B2 | MOVL | LOC STATUS, STATUS | : | 3121 |
| 50 | 5A | DO | 002B5 | MOVL | STATUS, R0 | : | 3125 |
| | | 04 | 002B8 | RET | | : | 3126 |

: Routine Size: 697 bytes. Routine Base: \$CODE\$ + 0942

| | | |
|--------|------|---|
| : 2479 | 3127 | 1 |
| : 2480 | 3128 | 1 |

```

: 2482 3129 1 GLOBAL ROUTINE DEQ_MOUNT_LOCK : NOVALUE =
: 2483 3130 1
: 2484 3131 1 :++
: 2485 3132 1
: 2486 3133 1 FUNCTIONAL DESCRIPTION:
: 2487 3134 1
: 2488 3135 1 This routine dequeus the mount interlock, if it exists.
: 2489 3136 1
: 2490 3137 1 CALLING SEQUENCE:
: 2491 3138 1
: 2492 3139 1 KERNEL_CALL ( DEQ_MOUNT_LOCK );
: 2493 3140 1
: 2494 3141 1 This routine is called in kernel mode because the mount interlock
: 2495 3142 1 is taken out in kernel mode.
: 2496 3143 1
: 2497 3144 1 INPUT:
: 2498 3145 1
: 2499 3146 1 None.
: 2500 3147 1
: 2501 3148 1 OUTPUT:
: 2502 3149 1
: 2503 3150 1 None.
: 2504 3151 1
: 2505 3152 1 IMPLICIT INPUT:
: 2506 3153 1
: 2507 3154 1 LOCK_STATUS : Lock status block of the mount interlock
: 2508 3155 1
: 2509 3156 1 IMPLICIT OUTPUT:
: 2510 3157 1
: 2511 3158 1 None.
: 2512 3159 1
: 2513 3160 1 ROUTINE VALUE:
: 2514 3161 1
: 2515 3162 1 None.
: 2516 3163 1
: 2517 3164 1 SIDE EFFECTS:
: 2518 3165 1
: 2519 3166 1 Mount interlock released.
: 2520 3167 1
: 2521 3168 1 --
: 2522 3169 1
: 2523 3170 2 BEGIN
: 2524 3171 2
: 2525 3172 2 IF .LOCK_STATUS[1] NEQ 0 ! If mount lock exists,
: 2526 3173 2 THEN $DEQ (LKID = .LOCK_STATUS[1]); ! Release it
: 2527 3174 2
: 2528 3175 2
: 2529 3176 2 RETURN; ! Back to caller
: 2530 3177 1 END; ! End of routine DEQ_MOUNT_LOCK

```

```

50 00000000' 0000 0000
EF 00 00002
OD 13 00009

```

```

.ENTRY DEQ_MOUNT_LOCK, Save nothing
MOVL LOCK_STATUS+4, R0
BEQL 1$

```

```

: 3129
: 3172
:

```



```

2533 3179 1
2534 3180 1 ROUTINE WAIT_DELTA (N) : NOVALUE =
2535 3181 1
2536 3182 1 ++
2537 3183 1
2538 3184 1 FUNCTIONAL DESCRIPTION:
2539 3185 1
2540 3186 1 This routine goes into the waitfor state for a small period of
2541 3187 1 time. This wait period is introduced to give simultaneous mounts
2542 3188 1 a chance to get both the device lock and the mount interlock.
2543 3189 1 The amount of time spent in waitfor state is node-dependent (e.g.
2544 3190 1 based on the scssystemid). The wait time also varies from one
2545 3191 1 call to the next (with a small positive or negative bias).
2546 3192 1
2547 3193 1 CALLING SEQUENCE:
2548 3194 1
2549 3195 1 WAIT_DELTA (ARG1)
2550 3196 1
2551 3197 1 INPUT:
2552 3198 1
2553 3199 1 ARG1 : Number of times this routine has been called.
2554 3200 1
2555 3201 1 OUTPUT:
2556 3202 1
2557 3203 1 None.
2558 3204 1
2559 3205 1 IMPLICIT INPUT:
2560 3206 1
2561 3207 1 None.
2562 3208 1
2563 3209 1 IMPLICIT OUTPUT:
2564 3210 1
2565 3211 1 None.
2566 3212 1
2567 3213 1 ROUTINE VALUE:
2568 3214 1
2569 3215 1 None.
2570 3216 1
2571 3217 1 SIDE EFFECTS:
2572 3218 1
2573 3219 1 None.
2574 3220 1
2575 3221 1 --
2576 3222 1
2577 3223 2 BEGIN
2578 3224 2
2579 3225 2 OWN
2580 3226 2 SCSSYSID,
2581 3227 2 XDELTA,
2582 3228 2 BIAS,
2583 3229 2 GETS_ITMLST : BLOCK [(1+12)+4, BYTE] INITIAL
2584 3230 2 ( WORD (4)
2585 3231 2 WORD (SYS$SCSSYSTEMID),
2586 3232 2 LONG (SCSSYSID),
2587 3233 2 LONG (0),
2588 3234 2 LONG (0));
2589 3235 2 LOCAL

```

```

2590 3236 2 STATUS,
2591 3237 2 DELTA      : VECTOR [2, LONG] INITIAL (-1, -1);
2592 3238 2
2593 3239 2
2594 3240 2 Set up some initial values for the first call to this routine.
2595 3241 2
2596 3242 2 IF .N EQL 0
2597 3243 2 THEN
2598 3244 2 BEGIN
2599 3245 2
2600 3246 2   SCSSYSID = 0;
2601 3247 2
2602 3248 2   STATUS = $GETSYIW ( EFN      = MOUNT_EFN,
2603 3249 2                     ITMLST = GETS_ITMLST );
2604 3250 2
2605 3251 2
2606 3252 2   ; If the $GETSYI failed or scssystemid is zero, use a default value.
2607 3253 2
2608 3254 2
2609 3255 2   IF NOT .STATUS
2610 3256 2   OR .SCSSYSID EQL 0
2611 3257 2   THEN
2612 3258 2     SCSSYSID = 64;
2613 3259 2
2614 3260 2
2615 3261 2   ; Compute the initial delta time.
2616 3262 2
2617 3263 2   XDELTA = .(SCSSYSID)<0,7>;
2618 3264 2
2619 3265 2
2620 3266 2   ; Set up the bias. We set up a positive bias if the initial value
2621 3267 2   is "sufficiently" small. Otherwise, we set up a positive bias.
2622 3268 2
2623 3269 2   IF .XDELTA GEQ 64
2624 3270 2   THEN
2625 3271 2     BIAS = -1
2626 3272 2   ELSE
2627 3273 2     BIAS = +1;
2628 3274 2
2629 3275 2   END;
2630 3276 2
2631 3277 2
2632 3278 2   The actual delta is the previous delta plus the bias, i.e.
2633 3279 2   (previous_delta+bias) * 1 million * 100 nanosecond
2634 3280 2
2635 3281 2   This gives the range of
2636 3282 2
2637 3283 2     scssystemid<0,7> = 1+bias      .1 second + bias
2638 3284 2
2639 3285 2     scssystemid<0,7> = 128+bias   12.8 seconds + bias
2640 3286 2
2641 3287 2
2642 3288 2   The bias is + or - .1 second, depending on the previous delta time.
2643 3289 2   If delta is large, we set up a negative bias for the next iteration.
2644 3290 2   If delta is small, we set up a positive bias for the next iteration.
2645 3291 2
2646 3292 2

```

```

: 2647      3293 2 IF .XDELTA GEQ 128      ' Large xdelta, set negative bias
: 2648      3294 THEN
: 2649      3295     BIAS = - 1;
: 2650      3296
: 2651      3297 IF .XDELTA LEQ 10      ' Small xdelta, set positive bias
: 2652      3298 THEN
: 2653      3299     BIAS = + 1;
: 2654      3300
: 2655      3301 XDELTA = .XDELTA + .BIAS;      ! Compute new xdelta
: 2656      3302
: 2657      3303 DELTA [0] = .XDELTA * (-1 * 1000 * 1000);      ! Compute delta in 100 nanoseconds
: 2658      3304
: 2659      P 3305 STATUS = $SETIMR ( EFN = MOUNT_EFN,      ! Set timer
: 2660      3306                      DAYTIM = DELTA );
: 2661      3307
: 2662      3308 IF .STATUS
: 2663      3309 THEN
: 2664      3310     $WAITFR ( EFN = MOUNT_EFN );      ! Wait
: 2665      3311
: 2666      3312 RETURN;      ! Back to caller
: 2667      3313 1 END;      ! End of routine WAIT_DELTA

```

```

.PSECT $OWNS,NOEXE,2
00D0C SCSSYSID:
      00D10 XDELTA: .BLKB 4
      00D14 BIAS: .BLKB 4
0004 00D18 GETS_ITMLST:
      .WORD 4
      1065 00D1A .WORD 4197
00000000' 00D1C .ADDRESS SCSSYSID
00000000 00D20 .LONG 0
00000000 00D24 .LONG 0
      .EXTRN SYSS$SETIMR, SYSS$WAITFR
.PSECT $CODE$,NOWRT,2
0004 00000 WAIT_DELTA:
      .WORD Save R2 : 3180
      MOVAB XDELTA, R2
      SL3L2 #4, SP
      MNEGL #1, DELTA : 3223
      MNEGL #1, DELTA+4
      TSTL N : 3242
      BNEQ 4$
      FC A2 D4 00018 CLRL SCSSYSID : 3246
      7E 7C 0001B CLRQ -(SP) : 3249
      7E D4 0001D CLRL -(SP)
      08 A2 9F 0001F PUSHAB GETS_ITMLST
      7E 7C 00022 CLRQ -(SP)
      1A DD 00024 PUSHL #26
      00000000G 00 07 FB 00026 CALLS #7, SYSS$GETSYIW
      CS 50 E9 0002D BLBC STATUS, i$ : 3255

```


| | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|-----------|---------------------|-------------|--------------------------|-------------|------|------|
| 62 | FC | A2 | FC | A2 | 40 | FC | A2 | D5 00030 | TSTL | SCSSYSID | : | 3256 | |
| | | | | | | | 05 | 12 00033 | BNEQ | 2\$ | : | | |
| | | | | | | | 8F | 9A 00035 1\$: | MOVZBL | #64, SCSSYSID | : | 3258 | |
| | | | | | | | 00 | EF 0003A 2\$: | EXTZV | #0, #7, SCSSYSID, XDELTA | : | 3263 | |
| | | | | | | | 3F | 62 D1 00040 | CMPL | XDELTA, #63 | : | 3269 | |
| | | | | | | | 06 | 15 00043 | BLEQ | 3\$ | : | | |
| | | | | | | | 04 | A2 01 CE 00045 | MNEGL | #1, BIAS | : | 3271 | |
| | | | | | | | 04 | 11 00049 | BRB | 4\$ | : | | |
| | | | | | | | 04 | A2 01 D0 0004B 3\$: | MOVL | #1, BIAS | : | 3273 | |
| | | | | | | | 00000080 | 8F 62 D1 0004F 4\$: | CMPL | XDELTA, #128 | : | 3293 | |
| | | | | | | | 04 | 19 00056 | BLSS | 5\$ | : | | |
| | | | | | | | 04 | A2 01 CE 00058 | MNEGL | #1, BIAS | : | 3295 | |
| | | | | | | | 0A | 62 D1 0005C 5\$: | CMPL | XDELTA, #10 | : | 3297 | |
| | | | | | | | 04 | 14 0005F | BGTR | 6\$ | : | | |
| | | | | | | | 04 | A2 01 D0 0C061 | MOVL | #1, BIAS | : | 3299 | |
| | | | | | | | 04 | A2 C0 00065 6\$: | ADDL2 | BIAS, XDELTA | : | 3301 | |
| | | | | | | | 6E | 62 FFF0BD C0 | MULL3 | #-1000000, XDELTA, DELTA | : | 3303 | |
| | | | | | | | | 8F | C5 00069 | | : | 3306 | |
| | | | | | | | | 7E | 7C 00071 | CLRQ | -(SP) | : | |
| | | | | | | | 08 | AE 9F 00073 | PUSHAB | DELTA | : | | |
| | | | | | | | | 1A | DD 00076 | PUSHL | #26 | : | |
| | | | | | | | 00000000G | 00 04 FB 00078 | CALLS | #4, SYSS\$SETIMR | : | | |
| | | | | | | | | 09 | 50 E9 0007F | BLBC | STATUS, 7\$ | : | 3308 |
| | | | | | | | | 1A | DD 00082 | PUSHL | #26 | : | 3310 |
| | | | | | | | 00000000G | 00 01 FB 00084 | CALLS | #1, SYSS\$WAITFR | : | | |
| | | | | | | | | 04 | 0008B 7\$: | RET | : | 3313 | |

: Routine Size: 140 bytes, Routine Base: \$CODE\$ + 0C14

```
: 2668      3314  1
: 2669      3315  1 END
: 2670      3316  0 ELUDOM
```

.EXTRN LIB\$SIGNAL, LIB\$STOP

PSECT SUMMARY

| Name | Bytes | Attributes |
|-----------|-------|--|
| \$GLOBALS | 1672 | NOVEC, WRT, RD, NOEXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2) |
| \$CODE\$ | 3232 | NOVEC, NOWRT, RD, EXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2) |
| \$SPLIT\$ | 96 | NOVEC, NOWRT, RD, NOEXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2) |
| \$OWNS | 3368 | NOVEC, WRT, RD, NOEXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2) |

Library Statistics

| File | ----- Symbols ----- | | Pages Mapped | Processing Time |
|---------------------------------|---------------------|----------------|--------------|-----------------|
| | Total | Loaded Percent | | |
| _\$255\$DUA28:[SYSLIB]LIB.L32;1 | 18619 | 122 0 | 1000 | 00:01.8 |

MPCLRPFM LIS

MPAST LIS

MP

MP MAP

MP MDL

TRNLOG LIS

MPCMOD LIS

MPMACROS MAR