

```

*****
10877 Sun May 4 18:28:35 2014
new/usr/src/cmd/Makefile
4833 Remove volrmmount
4845 rm(u)mount don't always print mount/unmount errors
4846 HAL partition names don't match real partition names
Reviewed by: Dan McDonald <danmcd@omniti.com>
Reviewed by: Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
*****
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #

22 # Copyright (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
23 # Copyright 2010 Nexenta Systems, Inc. All rights reserved.
24 # Copyright (c) 2012 Joyent, Inc. All rights reserved.
25 # Copyright (c) 2012 by Delphix. All rights reserved.
26 # Copyright (c) 2013 DEY Storage Systems, Inc. All rights reserved.
27 # Copyright 2014 Andrew Stormont.
28 #endif /* ! codereview */

30 include ../Makefile.master

32 #
33 # Note that the commands 'agents', 'lp', 'perl', and 'man' are first in
34 # the list, violating alphabetical order. This is because they are very
35 # long-running and should be given the most wall-clock time for a
36 # parallel build.
37 #
38 # Commands in the FIRST_SUBDIRS list are built before starting the build
39 # of other commands. Currently this includes only 'isaexec' and
40 # 'platexec'. This is necessary because $(ROOT)/usr/lib/isaexec or
41 # $(ROOT)/usr/lib/platexec must exist when some other commands are built
42 # because their 'make install' creates a hard link to one of them.
43 #
44 # Commands are listed one per line so that TeamWare can auto-merge most
45 # changes.
46 #

48 FIRST_SUBDIRS= \
49 isaexec \
50 platexec

52 COMMON_SUBDIRS= \
53 allocate \
54 availdevs \
55 lp \
56 perl \
57 man

```

```

58 Adm \
59 abi \
60 adbgen \
61 acct \
62 acctadm \
63 arch \
64 asa \
65 ast \
66 audio \
67 auths \
68 autopush \
69 avs \
70 awk \
71 awk_xpg4 \
72 backup \
73 banner \
74 bart \
75 basename \
76 bc \
77 bdiff \
78 beadm \
79 bfs \
80 bnu \
81 boot \
82 busstat \
83 cal \
84 calendar \
85 captinfo \
86 cat \
87 cdrw \
88 cfdm \
89 checkeq \
90 checknr \
91 chgrp \
92 chmod \
93 chown \
94 chroot \
95 clear \
96 clinfo \
97 cmd-crypto \
98 cmd-inet \
99 col \
100 compress \
101 consadm \
102 coreadm \
103 cpio \
104 cpc \
105 cron \
106 crypt \
107 csh \
108 csplit \
109 ctrun \
110 ctstat \
111 ctwatch \
112 datadm \
113 date \
114 dc \
115 dd \
116 deroff \
117 devfsadm \
118 syseventd \
119 devctl \
120 devinfo \
121 devmgmt \
122 devprop \
123 dfs.cmds \

```

```

124 diff \
125 diff3 \
126 diffmk \
127 dircmp \
128 dirname \
129 dis \
130 diskmgtd \
131 dispadmin \
132 dladm \
133 dlstat \
134 dmesg \
135 dodatadm \
136 dtrace \
137 du \
138 dumpadm \
139 dumpcs \
140 echo \
141 ed \
142 eeeprom \
143 egrep \
144 eject \
145 emul64ioctl \
146 enhance \
147 env \
148 eqn \
149 expand \
150 expr \
151 exstr \
152 factor \
153 false \
154 fcinfo \
155 fcoesvc \
156 fdetach \
157 fdformat \
158 fdisk \
159 filesync \
160 fgrep \
161 file \
162 filebench \
163 find \
164 flowadm \
165 flowstat \
166 fm \
167 fmt \
168 fmthard \
169 fmtmsg \
170 fold \
171 format \
172 fs.d \
173 fstyp \
174 fuser \
175 fwflash \
176 gcore \
177 gencat \
178 geniconvtbl \
179 genmsg \
180 getconf \
181 getdevpolicy \
182 getent \
183 getfacl \
184 getmajor \
185 getopt \
186 gettext \
187 gettxt \
188 grep \
189 grep_xpg4 \

```

```

190 groups \
191 grpck \
192 gss \
193 hal \
194 halt \
195 head \
196 hostid \
197 hostname \
198 hotplug \
199 hotplugd \
200 hwdata \
201 ibd_upgrade \
202 id \
203 idmap \
204 infocmp \
205 init \
206 initpkg \
207 install.d \
208 intrd \
209 intrstat \
210 ipcrm \
211 ipcs \
212 ipdadm \
213 ipf \
214 isainfo \
215 isalist \
216 itutools \
217 iscsiadm \
218 iscsid \
219 iscsitsvc \
220 isns \
221 itadm \
222 java \
223 kbd \
224 keyserv \
225 killall \
226 krb5 \
227 ksh \
228 kvmstat \
229 last \
230 lastcomm \
231 latencytop \
232 ldap \
233 ldapcachemgr \
234 lgrpinfo \
235 line \
236 link \
237 dlmgmt \
238 listen \
239 loadkeys \
240 locale \
241 localedef \
242 lockstat \
243 locator \
244 lofiadm \
245 logadm \
246 logger \
247 login \
248 logins \
249 look \
250 ls \
251 luxadm \
252 lvm \
253 mach \
254 machid \
255 mail \

```

```

256 mailx \
257 makekey \
258 mdb \
259 mesg \
260 mkdir \
261 mkfifo \
262 mkfile \
263 mkmsgs \
264 mknod \
265 mkpwdict \
266 mktemp \
267 modload \
268 more \
269 mpathadm \
270 msgfmt \
271 msgid \
272 mt \
273 mv \
274 mvdir \
275 ndmpadm \
276 ndmpd \
277 ndmpstat \
278 netadm \
279 netfiles \
280 newform \
281 newgrp \
282 news \
283 newtask \
284 nice \
285 nl \
286 nlsadmin \
287 nohup \
288 nsadmin \
289 nscd \
290 oamuser \
291 oawk \
292 od \
293 pack \
294 pagesize \
295 passmgmt \
296 passwd \
297 pathchk \
298 pbind \
299 pcidr \
300 pcitool \
301 pfexec \
302 pfexecd \
303 pginfo \
304 pgstat \
305 pgrep \
306 picl \
307 plimit \
308 policykit \
309 pools \
310 power \
311 powertop \
312 ppgsz \
313 pg \
314 plockstat \
315 pr \
316 prctl \
317 print \
318 printf \
319 priocntl \
320 profiles \
321 projadd \

```

```

322 projects \
323 prstat \
324 prtconf \
325 prtdiag \
326 prtvtoc \
327 ps \
328 psradm \
329 psrinfo \
330 psrset \
331 ptools \
332 pwck \
333 pwconv \
334 pwd \
335 pyzfs \
336 raidctl \
337 ramdiskadm \
338 rcap \
339 rcm_daemon \
340 rctladm \
341 refer \
342 regcmp \
343 renice \
344 rexd \
345 rm \
346 rmdir \
347 rmformat \
348 rmmount \
349 rmt \
350 rmvolmgr \
351 roles \
352 rpcbind \
353 rpcgen \
354 rpcinfo \
355 rpcsvc \
356 runat \
357 sa \
358 saf \
359 sasinfo \
360 savecore \
361 sbdadm \
362 script \
363 scsi \
364 sdiff \
365 sdpadm \
366 sed \
367 sendmail \
368 setfacl \
369 setmnt \
370 setpgrp \
371 setuname \
372 sgs \
373 sh \
374 shcomp \
375 smbios \
376 smbstrv \
377 smserverd \
378 soelim \
379 sort \
380 spell \
381 split \
382 sqlite \
383 srchtxt \
384 srptadm \
385 srptsvc \
386 ssh \
387 stat \

```

```

388     stmfadm      \
389     stmfproxy   \
390     stmfsvc     \
391     stmsboot    \
392     streams     \
393     strings     \
394     su          \
395     sulogin     \
396     sunpc       \
397     svc         \
398     svr4pkg     \
399     swap        \
400     sync        \
401     sysdef      \
402     syseventadm \
403     syslogd     \
404     tabs        \
405     tail        \
406     tar         \
407     tbl         \
408     tcopy       \
409     tcpd        \
410     terminfo    \
411     th_tools    \
412     tic         \
413     time        \
414     tip         \
415     tnf         \
416     touch       \
417     tput        \
418     tr          \
419     trapstat    \
420     troff       \
421     true        \
422     truss       \
423     tsol        \
424     tty         \
425     ttymon      \
426     tzreload    \
427     uadmin      \
428     ul          \
429     uname       \
430     units       \
431     unlink      \
432     unpack      \
433     userattr    \
434     users       \
435     utmp_update \
436     utmpd       \
437     valtools    \
438     vgrind      \
439     vi          \
440     volcheck    \
27     volrmmount \
441     vrrpadm     \
442     vscan       \
443     vt          \
444     w           \
445     wall        \
446     which       \
447     who         \
448     whodo       \
449     wracct      \
450     write       \
451     wusbadm     \
452     xargs       \

```

```

453     xstr        \
454     yes         \
455     ypcmd       \
456     yppasswd    \
457     zdb         \
458     zdump       \
459     zfs         \
460     zhack       \
461     zic         \
462     zinject     \
463     zlogin      \
464     zoneadm     \
465     zoneadm     \
466     zonecfg     \
467     zonename    \
468     zpool       \
469     zlook       \
470     zonestat    \
471     zstreamdump \
472     ztest       \
473     \
474     i386_SUBDIRS= \
475     acpihp       \
476     addbadsec   \
477     biosdev     \
478     diskscan    \
479     lms          \
480     ntfsprogs   \
481     parted      \
482     rtc         \
483     ucodeadm    \
484     xvm         \
485     \
486     sparc_SUBDIRS= \
487     cvcd        \
488     dcs         \
489     device_remap \
490     drd         \
491     fruadm      \
492     ldmad       \
493     oplhpd      \
494     prtdscp     \
495     prtfru      \
496     scadm       \
497     sockmd      \
498     sf880drd   \
499     virtinfo    \
500     vntsd       \
501     \
502     #
503     # Commands that are messaged. Note that 'lp' and 'man' come first
504     # (see previous comment about 'lp' and 'man').
505     #
506     MSGSUBDIRS= \
507     lp          \
508     man         \
509     abi         \
510     acctadm     \
511     allocate    \
512     asa         \
513     audio       \
514     audit       \
515     auditconfig \
516     auditd      \
517     auditrecord \
518     auditset    \

```

new/usr/src/cmd/Makefile

```

519     auths          //
520     autopush      //
521     avs           //
522     awk           //
523     awk_xpg4     //
524     backup       //
525     banner       //
526     bart         //
527     basename     //
528     beadm        //
529     bnu          //
530     busstat      //
531     cal          //
532     cat          //
533     cdrw        //
534     cfgadm       //
535     checkeq      //
536     checknr      //
537     chgrp        //
538     chmod        //
539     chown        //
540     cmd-crypto   //
541     cmd-inet     //
542     col          //
543     compress     //
544     consadm      //
545     coreadm      //
546     cpio         //
547     cpc          //
548     cron         //
549     csh          //
550     csplit       //
551     ctrun        //
552     ctstat       //
553     ctwatch      //
554     datadm       //
555     date         //
556     dc           //
557     dcs          //
558     dd           //
559     deroff       //
560     devfsadm     //
561     dfs.cmds     //
562     diff         //
563     diffmk       //
564     dladm        //
565     dlstat       //
566     du           //
567     dumpcs       //
568     ed           //
569     eject        //
570     env          //
571     eqn          //
572     expand        //
573     expr         //
574     fcinfo       //
575     fgrep        //
576     file         //
577     filesync     //
578     find         //
579     flowadm      //
580     flowstat     //
581     fm           //
582     fold         //
583     fs.d         //
584     fwflash      //

```

9

new/usr/src/cmd/Makefile

```

585     geniconvtbl //
586     genmsg       //
587     getconf      //
588     getent       //
589     gettext      //
590     gettxt       //
591     grep         //
592     grep_xpg4   //
593     grpck        //
594     gss          //
595     halt         //
596     head         //
597     hostname     //
598     hotplug      //
599     id           //
600     idmap        //
601     isaexec      //
602     iscsiadm     //
603     iscsid       //
604     isns         //
605     itadm        //
606     kbd          //
607     krb5         //
608     ksh          //
609     last         //
610     ldap         //
611     ldapcachemgr //
612     lgrpinfo     //
613     locale       //
614     lofiadm      //
615     logadm       //
616     logger       //
617     logins       //
618     ls           //
619     luxadm       //
620     lvm          //
621     mailx        //
622     msg          //
623     mkdir        //
624     mkpwdict     //
625     mktemp       //
626     more         //
627     mpathadm     //
628     msgfmt       //
629     mv           //
630     ndmpadm      //
631     ndmpstat     //
632     newgrp       //
633     newtask      //
634     nice         //
635     nohup        //
636     oawk         //
637     pack         //
638     passwd       //
639     passmgmt     //
640     pathchk      //
641     pfexec       //
642     pg           //
643     pgrep        //
644     picl         //
645     pools        //
646     power        //
647     pr           //
648     praudit      //
649     print        //
650     profiles     //

```

10

```

651     projadd      \|
652     projects    \|
653     prstat      \|
654     prtdiag     \|
655     ps           \|
656     psrinfo     \|
657     ptools      \|
658     pwconv      \|
659     pwd         \|
660     pyzfs       \|
661     raidctl     \|
662     ramdiskadm  \|
663     rcap        \|
664     rcm_daemon  \|
665     refer       \|
666     regcmp      \|
667     renice      \|
668     roles       \|
669     rm          \|
670     rmdir       \|
671     rmformat    \|
672     rmmount     \|
673     rmvolmgr    \|
674     sasinfo     \|
675     sbdadm      \|
676     scadm       \|
677     script      \|
678     scsi        \|
679     sdiff       \|
680     sdpadm      \|
681     sgs         \|
682     sh          \|
683     shcomp      \|
684     smbstrv     \|
685     sort        \|
686     split       \|
687     srptadm     \|
688     ssh         \|
689     stat        \|
690     stmfadm     \|
691     stmsboot    \|
692     strings     \|
693     su          \|
694     svc         \|
695     svr4pkg     \|
696     swap        \|
697     syseventadm \|
698     syseventd  \|
699     tabs        \|
700     tar         \|
701     tbl         \|
702     time        \|
703     tnf         \|
704     touch       \|
705     tput        \|
706     troff       \|
707     tsol        \|
708     tty         \|
709     ttymon     \|
710     tzreload    \|
711     ul          \|
712     uname       \|
713     units       \|
714     unlink      \|
715     unpack      \|
716     userattr    \|

```

```

717     valtools    \|
718     vgrind      \|
719     vi          \|
720     volcheck    \|
308     volrmmount \|
721     vrrpadm     \|
722     vscan       \|
723     w           \|
724     who         \|
725     whodo       \|
726     wracct      \|
727     write       \|
728     wusbadm     \|
729     xargs       \|
730     yppasswd    \|
731     zdump       \|
732     zfs         \|
733     zic         \|
734     zlogin      \|
735     zoneadm     \|
736     zoneadmmd  \|
737     zonecfg     \|
738     zonename    \|
739     zpool       \|
740     zonestat    \|

742     sparc_MSGSUBDIRS= \|
743     fruadm       \|
744     prtddscp    \|
745     prtfru      \|
746     virtinfo    \|
747     vntsd       \|

749     i386_MSGSUBDIRS= \|
750     ucodeadm    \|

752 #
753 # commands that use dcgettext for localized time, LC_TIME
754 #
755     DCSUBDIRS=    \|
756     cal          \|
757     cfgadm       \|
758     diff         \|
759     ls           \|
760     pr           \|
761     ps           \|
762     tar          \|
763     w           \|
764     who         \|
765     whodo       \|
766     write       \|

768 #
769 # commands that belong only to audit.
770 #
771     AUDITSUBDIRS= \|
772     amt          \|
773     audit        \|
774     audit_warn   \|
775     auditconfig  \|
776     auditd       \|
777     auditrecord  \|
778     auditreduce  \|
779     auditset     \|
780     auditstat    \|
781     praudit     \|

```

```
783 #
784 # commands not owned by the systems group
785 #
786 BWOSDIRS=

789 all :=          TARGET = all
790 install :=     TARGET = install
791 clean :=      TARGET = clean
792 clobber :=   TARGET = clobber
793 lint :=      TARGET = lint
794 _msg :=     TARGET = _msg
795 _dc :=      TARGET = _dc

797 .KEEP_STATE:

799 SUBDIRS = $(COMMON_SUBDIRS) $( $(MACH)_SUBDIRS )

801 .PARALLEL:      $(BWOSDIRS) $(SUBDIRS) $(MSGSUBDIRS) $(AUDITSUBDIRS)

803 all install clean clobber lint: $(FIRST_SUBDIRS) .WAIT $(SUBDIRS) \
804     $(AUDITSUBDIRS)

806 #
807 # Manifests cannot be checked in parallel, because we are using
808 # the global repository that is in $(SRC)/cmd/svc/seed/global.db.
809 # For this reason, to avoid .PARALLEL and .NO_PARALLEL conflicts,
810 # we spawn off a sub-make to perform the non-parallel 'make check'
811 #
812 check:
813     $(MAKE) -f Makefile.check check

815 #
816 # The .WAIT directive works around an apparent bug in parallel make.
817 # Evidently make was getting the target _msg vs. _dc confused under
818 # some level of parallelization, causing some of the _dc objects
819 # not to be built.
820 #
821 _msg: $(MSGSUBDIRS) $( $(MACH)_MSGSUBDIRS ) .WAIT _dc

823 _dc: $(DCSUBDIRS)

825 #
826 # Dependencies
827 #
828 fs.d: fstyp
829 ksh:  shcomp isaexec
830 mdb:  terminfo
831 print: lp

833 $(FIRST_SUBDIRS) $(BWOSDIRS) $(SUBDIRS) $(AUDITSUBDIRS): FRC
834     @if [ -f $@/Makefile ]; then \
835         cd $@; pwd; $(MAKE) $(TARGET); \
836     else \
837         true; \
838     fi

840 FRC:
```

```

*****
52155 Sun May 4 18:28:35 2014
new/usr/src/cmd/hal/hald/solaris/devinfo_storage.c
4833 Remove volrmmount
4845 rm(u)mount don't always print mount/unmount errors
4846 HAL partition names don't match real partition names
Reviewed by: Dan McDonald <danmcd@omniti.com>
Reviewed by: Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
*****
1 /*****
2 *
3 * devinfo_storage.c : storage devices
4 *
5 * Copyright (c) 2006, 2010, Oracle and/or its affiliates. All rights reserved.
6 * Copyright 2013 Garrett D'Amore <garrett@damore.org>
7 * Copyright 2014 Andrew Stormont.
8 #endif /* ! codereview */
9 *
10 * Licensed under the Academic Free License version 2.1
11 *
12 *****/

14 #ifndef HAVE_CONFIG_H
15 # include <config.h>
16 #endif

18 #include <stdio.h>
19 #include <string.h>
20 #include <strings.h>
21 #include <ctype.h>
22 #include <libdevinfo.h>
23 #include <sys/types.h>
24 #include <sys/mkdev.h>
25 #include <sys/stat.h>
26 #include <sys/mntent.h>
27 #include <sys/mnttab.h>

29 #include "../osspec.h"
30 #include "../logger.h"
31 #include "../hald.h"
32 #include "../hald_dbus.h"
33 #include "../device_info.h"
34 #include "../util.h"
35 #include "../hald_runner.h"
36 #include "hotplug.h"
37 #include "devinfo.h"
38 #include "devinfo_misc.h"
39 #include "devinfo_storage.h"
40 #include "osspec_solaris.h"

42 #ifdef sparc
43 #define WHOLE_DISK "s2"
44 #else
45 #define WHOLE_DISK "p0"
46 #endif

48 /* some devices, especially CDROMS, may take a while to be probed (values in ms)
49 #define DEVINFO_PROBE_STORAGE_TIMEOUT 60000
50 #define DEVINFO_PROBE_VOLUME_TIMEOUT 60000

52 typedef struct devinfo_storage_minor {
53     char *devpath;
54     char *devlink;
55     char *slice;
56     dev_t dev;
57     int dosnum; /* dos disk number or -1 */

```

```

58 } devinfo_storage_minor_t;

60 HalDevice *devinfo_ide_add(HalDevice *parent, di_node_t node, char *devfs_path,
61 static HalDevice *devinfo_ide_host_add(HalDevice *parent, di_node_t node, char *
62 static HalDevice *devinfo_ide_device_add(HalDevice *parent, di_node_t node, char
63 static HalDevice *devinfo_ide_storage_add(HalDevice *parent, di_node_t node, cha
64 HalDevice *devinfo_scsi_add(HalDevice *parent, di_node_t node, char *devfs_path,
65 static HalDevice *devinfo_scsi_storage_add(HalDevice *parent, di_node_t node, ch
66 HalDevice *devinfo_blkdev_add(HalDevice *parent, di_node_t node, char *devfs_pat
67 static HalDevice *devinfo_blkdev_storage_add(HalDevice *parent, di_node_t node,
68 HalDevice *devinfo_floppy_add(HalDevice *parent, di_node_t node, char *devfs_pat
69 static void devinfo_floppy_add_volume(HalDevice *parent, di_node_t node);
70 static HalDevice *devinfo_lofi_add(HalDevice *parent, di_node_t node, char *devf
71 static void devinfo_lofi_add_minor(HalDevice *parent, di_node_t node, char *mino
72 static void devinfo_storage_minors(HalDevice *parent, di_node_t node, gchar *dev
73 static struct devinfo_storage_minor *devinfo_storage_new_minor(char *maindev_pat
74     char *devlink, dev_t dev, int dosnum);
75 static void devinfo_storage_free_minor(struct devinfo_storage_minor *m);
76 HalDevice *devinfo_volume_add(HalDevice *parent, di_node_t node, devinfo_storage
77 static void devinfo_volume_preprobing_done(HalDevice *d, gpointer userdata1, gpo
78 static void devinfo_volume_hotplug_begin_add (HalDevice *d, HalDevice *parent, D
79 static void devinfo_storage_hotplug_begin_add (HalDevice *d, HalDevice *parent,
80 static void devinfo_storage_probing_done (HalDevice *d, guint32 exit_type, gint
81 const gchar *devinfo_volume_get_prober (HalDevice *d, int *timeout);
82 const gchar *devinfo_storage_get_prober (HalDevice *d, int *timeout);

84 static char *devinfo_scsi_dtype2str(int dtype);
85 static char *devinfo_volume_get_slice_name (char *devlink);
86 static gboolean dos_to_dev(char *path, char **devpath, int *partnum);
87 static gboolean is_dos_path(char *path, int *partnum);

89 static void devinfo_storage_set_nicknames (HalDevice *d);

91 DevinfoDevHandler devinfo_ide_handler = {
92     devinfo_ide_add,
93     NULL,
94     NULL,
95     NULL,
96     NULL,
97     NULL
98 };
99 DevinfoDevHandler devinfo_scsi_handler = {
100     devinfo_scsi_add,
101     NULL,
102     NULL,
103     NULL,
104     NULL,
105     NULL
106 };
107 DevinfoDevHandler devinfo_blkdev_handler = {
108     devinfo_blkdev_add,
109     NULL,
110     NULL,
111     NULL,
112     NULL,
113     NULL
114 };
115 DevinfoDevHandler devinfo_floppy_handler = {
116     devinfo_floppy_add,
117     NULL,
118     NULL,
119     NULL,
120     NULL,
121     NULL
122 };
123 DevinfoDevHandler devinfo_lofi_handler = {

```



```

124     devinfo_lofi_add,
125     NULL,
126     NULL,
127     NULL,
128     NULL,
129     NULL
130 };
131 DevinfoDevHandler devinfo_storage_handler = {
132     NULL,
133     NULL,
134     devinfo_storage_hotplug_begin_add,
135     NULL,
136     devinfo_storage_probing_done,
137     devinfo_storage_get_prober
138 };
139 DevinfoDevHandler devinfo_volume_handler = {
140     NULL,
141     NULL,
142     devinfo_volume_hotplug_begin_add,
143     NULL,
144     NULL,
145     devinfo_volume_get_prober
146 };
147
148 /* IDE */
149
150 HalDevice *
151 devinfo_ide_add(HalDevice *parent, di_node_t node, char *devfs_path, char *device)
152 {
153     char *s;
154
155     if ((device_type != NULL) && (strcmp(device_type, "ide") == 0)) {
156         return (devinfo_ide_host_add(parent, node, devfs_path));
157     }
158
159     if ((di_prop_lookup_strings (DDI_DEV_T_ANY, node, "class", &s) > 0) &&
160         (strcmp (s, "dada") == 0)) {
161         return (devinfo_ide_device_add(parent, node, devfs_path));
162     }
163
164     return (NULL);
165 }
166
167 static HalDevice *
168 devinfo_ide_host_add(HalDevice *parent, di_node_t node, char *devfs_path)
169 {
170     HalDevice *d;
171
172     d = hal_device_new ();
173
174     devinfo_set_default_properties (d, parent, node, devfs_path);
175     hal_device_property_set_string (d, "info.product", "IDE host controller");
176     hal_device_property_set_string (d, "info.subsystem", "ide_host");
177     hal_device_property_set_int (d, "ide_host.number", 0); /* XXX */
178
179     devinfo_add_enqueue (d, devfs_path, &devinfo_ide_handler);
180
181     return (d);
182 }
183
184 static HalDevice *
185 devinfo_ide_device_add(HalDevice *parent, di_node_t node, char *devfs_path)
186 {
187     HalDevice *d;
188
189     d = hal_device_new();

```

```

191     devinfo_set_default_properties (d, parent, node, devfs_path);
192     hal_device_property_set_string (parent, "info.product", "IDE device");
193     hal_device_property_set_string (parent, "info.subsystem", "ide");
194     hal_device_property_set_int (parent, "ide.host", 0); /* XXX */
195     hal_device_property_set_int (parent, "ide.channel", 0);
196
197     devinfo_add_enqueue (d, devfs_path, &devinfo_ide_handler);
198
199     return (devinfo_ide_storage_add (d, node, devfs_path));
200 }
201
202 static HalDevice *
203 devinfo_ide_storage_add(HalDevice *parent, di_node_t node, char *devfs_path)
204 {
205     HalDevice *d;
206     char *s;
207     int *i;
208     char *driver_name;
209     char udi[HAL_PATH_MAX];
210
211     if ((driver_name = di_driver_name (node)) == NULL) {
212         return (NULL);
213     }
214
215     d = hal_device_new ();
216
217     devinfo_set_default_properties (d, parent, node, devfs_path);
218     hal_device_property_set_string (d, "info.category", "storage");
219
220     hal_util_compute_udi (hald_get_gdl (), udi, sizeof (udi),
221                          "%s/%s%d", hal_device_get_udi (parent), driver_name, di_instance);
222     hal_device_set_udi (d, udi);
223     hal_device_property_set_string (d, "info.udi", udi);
224     PROP_STR(d, node, s, "devid", "info.product");
225
226     hal_device_add_capability (d, "storage");
227     hal_device_property_set_string (d, "storage.bus", "ide");
228     hal_device_property_set_int (d, "storage.lun", 0);
229     hal_device_property_set_string (d, "storage.drive_type", "disk");
230
231     PROP_BOOL(d, node, i, "hotpluggable", "storage.hotpluggable");
232     PROP_BOOL(d, node, i, "removable-media", "storage.removable");
233
234     hal_device_property_set_bool (d, "storage.media_check_enabled", FALSE);
235
236     /* XXX */
237     hal_device_property_set_bool (d, "storage.requires_eject", FALSE);
238
239     hal_device_add_capability (d, "block");
240
241     devinfo_storage_minors (d, node, (char *)devfs_path, FALSE);
242
243     return (d);
244 }
245
246 /* SCSI */
247
248 HalDevice *
249 devinfo_scsi_add(HalDevice *parent, di_node_t node, char *devfs_path, char *device)
250 {
251     int *i;
252     char *driver_name;
253     HalDevice *d;
254     char udi[HAL_PATH_MAX];

```

```

256 driver_name = di_driver_name (node);
257 if ((driver_name == NULL) || (strcmp (driver_name, "sd") != 0)) {
258     return (NULL);
259 }
261 d = hal_device_new ();
263 devinfo_set_default_properties (d, parent, node, devfs_path);
264 hal_device_property_set_string (d, "info.subsystem", "scsi");
266 hal_util_compute_udi (hald_get_gdl (), udi, sizeof (udi),
267     "%s/%s%d", hal_device_get_udi (parent), di_node_name(node), di_i
268 hal_device_set_udi (d, udi);
269 hal_device_property_set_string (d, "info.udi", udi);
271 hal_device_property_set_int (d, "scsi.host",
272     hal_device_property_get_int (parent, "scsi_host.host"));
273 hal_device_property_set_int (d, "scsi.bus", 0);
274 PROP_INT(d, node, i, "target", "scsi.target");
275 PROP_INT(d, node, i, "lun", "scsi.lun");
276 hal_device_property_set_string (d, "info.product", "SCSI Device");
278 devinfo_add_enqueue (d, devfs_path, &devinfo_scsi_handler);
280 return (devinfo_scsi_storage_add (d, node, devfs_path));
281 }
283 static HalDevice *
284 devinfo_scsi_storage_add(HalDevice *parent, di_node_t node, char *devfs_path)
285 {
286     HalDevice *d;
287     int *i;
288     char *s;
289     char udi[HAL_PATH_MAX];
291     d = hal_device_new ();
293 devinfo_set_default_properties (d, parent, node, devfs_path);
294 hal_device_property_set_string (d, "info.category", "storage");
296 hal_util_compute_udi (hald_get_gdl (), udi, sizeof (udi),
297     "%s/sd%d", hal_device_get_udi (parent), di_instance (node));
298 hal_device_set_udi (d, udi);
299 hal_device_property_set_string (d, "info.udi", udi);
300 PROP_STR(d, node, s, "inquiry-product-id", "info.product");
302 hal_device_add_capability (d, "storage");
304 hal_device_property_set_int (d, "storage.lun",
305     hal_device_property_get_int (parent, "scsi.lun"));
306 PROP_BOOL(d, node, i, "hotpluggable", "storage.hotpluggable");
307 PROP_BOOL(d, node, i, "removable-media", "storage.removable");
308 hal_device_property_set_bool (d, "storage.requires_eject", FALSE);
310 /*
311  * We have to enable polling not only for drives with removable media,
312  * but also for hotpluggable devices, because when a disk is
313  * unplugged while busy/mounted, there is not sysevent generated.
314  * Instead, the HBA driver (scsa2usb, scsa1394) will notify sd driver
315  * and the latter will report DKIO_DEV_GONE via DKIOCSTATE ioctl.
316  * So we have to enable media check so that hald-addon-storage notices
317  * the "device gone" condition and unmounts all associated volumes.
318  */
319 hal_device_property_set_bool (d, "storage.media_check_enabled",
320     ((di_prop_lookup_ints(DDI_DEV_T_ANY, node, "removable-media", &i) >=
321     (di_prop_lookup_ints(DDI_DEV_T_ANY, node, "hotpluggable", &i) >= 0))

```

```

323     if (di_prop_lookup_ints(DDI_DEV_T_ANY, node, "inquiry-device-type",
324         &i) > 0) {
325         s = devinfo_scsi_dtype2str (*i);
326         hal_device_property_set_string (d, "storage.drive_type", s);
328         if (strcmp (s, "cdrom") == 0) {
329             hal_device_add_capability (d, "storage.cdrom");
330             hal_device_property_set_bool (d, "storage.no_partitions_
331                 hal_device_property_set_bool (d, "storage.requires_eject
332         }
333     }
335 hal_device_add_capability (d, "block");
337 devinfo_storage_minors (d, node, devfs_path, FALSE);
339 return (d);
340 }
342 static char *
343 devinfo_scsi_dtype2str(int dtype)
344 {
345     char *dtype2str[] = {
346         "disk", /* DTYPE_DIRECT 0x00 */
347         "tape", /* DTYPE_SEQUENTIAL 0x01 */
348         "printer", /* DTYPE_PRINTER 0x02 */
349         "processor", /* DTYPE_PROCESSOR 0x03 */
350         "worm", /* DTYPE_WORM 0x04 */
351         "cdrom", /* DTYPE_RODIRECT 0x05 */
352         "scanner", /* DTYPE_SCANNER 0x06 */
353         "cdrom", /* DTYPE_OPTICAL 0x07 */
354         "changer", /* DTYPE_CHANGER 0x08 */
355         "comm", /* DTYPE_COMM 0x09 */
356         "scsi", /* DTYPE_??? 0x0A */
357         "scsi", /* DTYPE_??? 0x0B */
358         "array_ctrl", /* DTYPE_ARRAY_CTRL 0x0C */
359         "esi", /* DTYPE_ESI 0x0D */
360         "disk", /* DTYPE_RBC 0x0E */
361     };
363     if (dtype < NELEM(dtype2str)) {
364         return (dtype2str[dtype]);
365     } else {
366         return ("scsi");
367     }
369 }
371 /* blkdev */
373 HalDevice *
374 devinfo_blkdev_add(HalDevice *parent, di_node_t node, char *devfs_path, char *de
375 {
376     int *i;
377     char *driver_name;
378     HalDevice *d;
379     char udi[HAL_PATH_MAX];
381     driver_name = di_driver_name (node);
382     if ((driver_name == NULL) || (strcmp (driver_name, "blkdev") != 0)) {
383         return (NULL);
384     }
386     d = hal_device_new ();

```

```

388 devinfo_set_default_properties (d, parent, node, devfs_path);
389 hal_device_property_set_string (d, "info.subsystem", "pseudo");

391 hal_util_compute_udi (hald_get_gdl (), udi, sizeof (udi),
392 "%s/%s%d", hal_device_get_udi (parent), di_node_name(node), di_i
393 hal_device_set_udi (d, udi);
394 hal_device_property_set_string (d, "info.udi", udi);
395 hal_device_property_set_string (d, "info.product", "Block Device");

397 devinfo_add_enqueue (d, devfs_path, &devinfo_blkdev_handler);

399 return (devinfo_blkdev_storage_add (d, node, devfs_path));
400 }

402 static HalDevice *
403 devinfo_blkdev_storage_add(HalDevice *parent, di_node_t node, char *devfs_path)
404 {
405     HalDevice *d;
406     char *driver_name;
407     int *i;
408     char *s;
409     char udi[HAL_PATH_MAX];

411     d = hal_device_new ();

413     devinfo_set_default_properties (d, parent, node, devfs_path);
414     hal_device_property_set_string (d, "info.category", "storage");

416     hal_util_compute_udi (hald_get_gdl (), udi, sizeof (udi),
417 "%s/blkdev%d", hal_device_get_udi (parent), di_instance (node));
418     hal_device_set_udi (d, udi);
419     hal_device_property_set_string (d, "info.udi", udi);

421     hal_device_add_capability (d, "storage");

423     hal_device_property_set_int (d, "storage.lun", 0);

425     PROP_BOOL(d, node, i, "hotpluggable", "storage.hotpluggable");
426     PROP_BOOL(d, node, i, "removable-media", "storage.removable");

428     hal_device_property_set_bool (d, "storage.requires_eject", FALSE);
429     hal_device_property_set_bool (d, "storage.media_check_enabled", TRUE);
430     hal_device_property_set_string (d, "storage.drive_type", "disk");

432     hal_device_add_capability (d, "block");

434     devinfo_storage_minors (d, node, devfs_path, FALSE);

436     return (d);
437 }

439 /* floppy */

441 HalDevice *
442 devinfo_floppy_add(HalDevice *parent, di_node_t node, char *devfs_path, char *de
443 {
444     char *driver_name;
445     char *raw;
446     char udi[HAL_PATH_MAX];
447     di_devlink_handle_t devlink_hdl;
448     int major;
449     di_minor_t minor;
450     dev_t dev;
451     HalDevice *d = NULL;
452     char *minor_path = NULL;
453     char *devlink = NULL;

```

```

455     driver_name = di_driver_name (node);
456     if ((driver_name == NULL) || (strcmp (driver_name, "fd") != 0)) {
457         return (NULL);
458     }

460     /*
461      * The only minor node we're interested in is /dev/diskette*
462      */
463     major = di_driver_major(node);
464     if ((devlink_hdl = di_devlink_init(NULL, 0)) == NULL) {
465         return (NULL);
466     }
467     minor = DI_MINOR_NIL;
468     while ((minor = di_minor_next(node, minor)) != DI_MINOR_NIL) {
469         dev = di_minor_devt(minor);
470         if ((major != major(dev)) ||
471             (di_minor_type(minor) != DDM_MINOR) ||
472             (di_minor_spectype(minor) != S_IFBLK) ||
473             ((minor_path = di_devfs_minor_path(minor)) == NULL)) {
474             continue;
475         }
476         if ((devlink = get_devlink(devlink_hdl, "diskette.+", minor_pat
477             break;
478         }
479         di_devfs_path_free (minor_path);
480         minor_path = NULL;
481         free(devlink);
482         devlink = NULL;
483     }
484     di_devlink_fini (&devlink_hdl);

486     if ((devlink == NULL) || (minor_path == NULL)) {
487         HAL_INFO (("floppy devlink not found %s", devfs_path));
488         goto out;
489     }

491     d = hal_device_new ();

493     devinfo_set_default_properties (d, parent, node, devfs_path);
494     hal_device_property_set_string (d, "info.category", "storage");
495     hal_device_add_capability (d, "storage");
496     hal_device_property_set_string (d, "storage.bus", "platform");
497     hal_device_property_set_bool (d, "storage.hotpluggable", FALSE);
498     hal_device_property_set_bool (d, "storage.removable", TRUE);
499     hal_device_property_set_bool (d, "storage.requires_eject", TRUE);
500     hal_device_property_set_bool (d, "storage.media_check_enabled", FALSE);
501     hal_device_property_set_string (d, "storage.drive_type", "floppy");

503     hal_device_add_capability (d, "block");
504     hal_device_property_set_bool (d, "block.is_volume", FALSE);
505     hal_device_property_set_int (d, "block.major", major(dev));
506     hal_device_property_set_int (d, "block.minor", minor(dev));
507     hal_device_property_set_string (d, "block.device", devlink);
508     raw = dsk_to_rdisk (devlink);
509     hal_device_property_set_string (d, "block.solaris.raw_device", raw);
510     free (raw);

512     devinfo_add_enqueue (d, devfs_path, &devinfo_storage_handler);

514     /* trigger initial probe-volume */
515     devinfo_floppy_add_volume(d, node);

517 out:
518     di_devfs_path_free (minor_path);
519     free(devlink);

```

```

521     return (d);
522 }

524 static void
525 devinfo_floppy_add_volume(HalDevice *parent, di_node_t node)
526 {
527     char *devlink;
528     char *devfs_path;
529     int minor, major;
530     dev_t dev;
531     struct devinfo_storage_minor *m;

533     devfs_path = (char *)hal_device_property_get_string (parent, "solaris.de
534 devlink = (char *)hal_device_property_get_string (parent, "block.device"
535 major = hal_device_property_get_int (parent, "block.major");
536 minor = hal_device_property_get_int (parent, "block.minor");
537 dev = makedev (major, minor);

539     m = devinfo_storage_new_minor (devfs_path, WHOLE_DISK, devlink, dev, -1)
540 devinfo_volume_add (parent, node, m);
541 devinfo_storage_free_minor (m);
542 }

544 /*
545  * After reprobng storage, reprobe its volumes.
546  */
547 static void
548 devinfo_floppy_rescan_probing_done (HalDevice *d, quint32 exit_type, gint return
549 char **error, gpointer userdatal, gpointer userdata2)
550 {
551     void *end_token = (void *) userdatal;
552     const char *devfs_path;
553     di_node_t node;
554     HalDevice *v;

556     if (!hal_device_property_get_bool (d, "storage.removable.media_available
557         HAL_INFO (("no floppy media", hal_device_get_udi (d)));

559     /* remove child (can only be single volume) */
560     if (((v = hal_device_store_match_key_value_string (hald_get_gdl(
561         "info.parent", hal_device_get_udi (d))) != NULL) &&
562         ((devfs_path = hal_device_property_get_string (v,
563             "solaris.devfs_path") != NULL)) {
564         devinfo_remove_enqueue ((char *)devfs_path, NULL);
565     }
566 } else {
567     HAL_INFO (("floppy media found", hal_device_get_udi (d)));

569     if ((devfs_path = hal_device_property_get_string(d, "solaris.dev
570         HAL_INFO (("no devfs_path", hal_device_get_udi (d)));
571         hotplug_event_process_queue ();
572         return;
573     }
574     if ((node = di_init (devfs_path, DINFOCOPYALL)) == DI_NODE_NIL) {
575         HAL_INFO (("di_init %s failed %d", devfs_path, errno));
576         hotplug_event_process_queue ();
577         return;
578     }

580     devinfo_floppy_add_volume (d, node);

582     di_fini (node);
583 }

585     hotplug_event_process_queue ();

```

```

586 }
587
588 /* lofi */

590 HalDevice *
591 devinfo_lofi_add(HalDevice *parent, di_node_t node, char *devfs_path, char *devi
592 {
593     return (devinfo_lofi_add_major(parent,node, devfs_path, device_type, FAL
594 }

596 HalDevice *
597 devinfo_lofi_add_major(HalDevice *parent, di_node_t node, char *devfs_path, char
598     gboolean rescan, HalDevice *lofi_d)
599 {
600     char *driver_name;
601     HalDevice *d = NULL;
602     char udi[HAL_PATH_MAX];
603     di_devlink_handle_t devlink_hdl;
604     int major;
605     di_minor_t minor;
606     dev_t dev;
607     char *minor_path = NULL;
608     char *devlink = NULL;

610     driver_name = di_driver_name (node);
611     if ((driver_name == NULL) || (strcmp (driver_name, "lofi") != 0)) {
612         return (NULL);
613     }

615     if (!rescan) {
616         d = hal_device_new ();

618         devinfo_set_default_properties (d, parent, node, devfs_path);
619         hal_device_property_set_string (d, "info.subsystem", "pseudo");

621         hal_util_compute_udi (hald_get_gdl (), udi, sizeof (udi),
622             "%s/%s%d", hal_device_get_udi (parent), di_node_name(nod
623         hal_device_set_udi (d, udi);
624         hal_device_property_set_string (d, "info.udi", udi);

626         devinfo_add_enqueue (d, devfs_path, &devinfo_lofi_handler);
627     } else {
628         d = lofi_d;
629     }

631     /*
632     * Unlike normal storage, as in devinfo_storage_minors(), where
633     * sd instance -> HAL storage, sd minor node -> HAL volume,
634     * lofi always has one instance, lofi minor -> HAL storage.
635     * lofi storage never has slices, but it can have
636     * embedded pcfs partitions that fstyp would recognize
637     */
638     major = di_driver_major(node);
639     if ((devlink_hdl = di_devlink_init(NULL, 0)) == NULL) {
640         return (d);
641     }

642     minor = DI_MINOR_NIL;
643     while ((minor = di_minor_next(node, minor)) != DI_MINOR_NIL) {
644         dev = di_minor_devt(minor);
645         if ((major != major(dev)) ||
646             (di_minor_type(minor) != DDM_MINOR) ||
647             (di_minor_spectype(minor) != S_IFBLK) ||
648             ((minor_path = di_devfs_minor_path(minor)) == NULL)) {
649             continue;
650         }
651         if ((devlink = get_devlink(devlink_hdl, NULL, minor_path)) == NU

```

```

652         di_devfs_path_free (minor_path);
653         continue;
654     }

656     if (!rescan ||
657         (hal_device_store_match_key_value_string (hald_get_gdl (),
658           "solaris.devfs_path", minor_path) == NULL)) {
659         devinfo_lofi_add_minor(d, node, minor_path, devlink, dev
660     }

662     di_devfs_path_free (minor_path);
663     free(devlink);
664 }
665 di_devlink_fini (&devlink_hdl);

667     return (d);
668 }

670 static void
671 devinfo_lofi_add_minor(HalDevice *parent, di_node_t node, char *minor_path, char
672 {
673     HalDevice *d;
674     char *raw;
675     char *doslink;
676     char dospath[64];
677     struct devinfo_storage_minor *m;
678     int i;

680     /* add storage */
681     d = hal_device_new ();

683     devinfo_set_default_properties (d, parent, node, minor_path);
684     hal_device_property_set_string (d, "info.category", "storage");
685     hal_device_add_capability (d, "storage");
686     hal_device_property_set_string (d, "storage.bus", "lofi");
687     hal_device_property_set_bool (d, "storage.hotpluggable", TRUE);
688     hal_device_property_set_bool (d, "storage.removable", FALSE);
689     hal_device_property_set_bool (d, "storage.requires_eject", FALSE);
690     hal_device_property_set_string (d, "storage.drive_type", "disk");
691     hal_device_add_capability (d, "block");
692     hal_device_property_set_int (d, "block.major", major(dev));
693     hal_device_property_set_int (d, "block.minor", minor(dev));
694     hal_device_property_set_string (d, "block.device", devlink);
695     raw = dsk_to_rdisk (devlink);
696     hal_device_property_set_string (d, "block.solaris.raw_device", raw);
697     free (raw);
698     hal_device_property_set_bool (d, "block.is_volume", FALSE);

700     devinfo_add_enqueue (d, minor_path, &devinfo_storage_handler);

702     /* add volumes: one on main device and a few pcfs candidates */
703     m = devinfo_storage_new_minor(minor_path, WHOLE_DISK, devlink, dev, -1);
704     devinfo_volume_add (d, node, m);
705     devinfo_storage_free_minor (m);

707     doslink = (char *)calloc (1, strlen (devlink) + sizeof ("pNN") + 1);
708     doslink = (char *)calloc (1, strlen (devlink) + sizeof (":NNN") + 1);
709     if (doslink != NULL) {
710         for (i = 1; i < 16; i++) {
711             sprintf(dospath, sizeof (dospath), "p%d", i);
712             sprintf(doslink, "%sp%d", devlink, i);
713             sprintf(dospath, sizeof (dospath), "WHOLE_DISK:%d", i);
714             sprintf(doslink, "%s:%d", devlink, i);
715             m = devinfo_storage_new_minor(minor_path, dospath, dosli
716             devinfo_volume_add (d, node, m);
717             devinfo_storage_free_minor (m);

```

```

715     }
716     free (doslink);
717 }
718 }

unchanged_portion_omitted

797 /*
798 * Storage minor nodes are potential "volume" objects.
799 * This function also completes building the parent object (main storage device)
800 */
801 static void
802 devinfo_storage_minors(HalDevice *parent, di_node_t node, gchar *devfs_path, gbo
803 {
804     di_devlink_handle_t devlink_hdl;
805     gboolean is_cdrom;
806     const char *whole_disk;
807     int major;
808     di_minor_t minor;
809     dev_t dev;
810     char *minor_path = NULL;
811     char *maindev_path = NULL;
812     char *devpath, *devlink;
813     int doslink_len;
814     char *doslink;
815     char dospath[64];
816     char *slice;
817     int pathlen;
818     int i;
819     char *raw;
820     boolean_t maindev_is_d0;
821     GQueue *mq;
822     HalDevice *volume;
823     struct devinfo_storage_minor *m;
824     struct devinfo_storage_minor *maindev = NULL;

826     /* for cdroms whole disk is always s2 */
827     is_cdrom = hal_device_has_capability (parent, "storage.cdrom");
828     whole_disk = is_cdrom ? "s2" : WHOLE_DISK;

830     major = di_driver_major(node);

832     /* the "whole disk" p0/s2/d0 node must come first in the hotplug queue
833     * so we put other minor nodes on the local queue and move to the
834     * hotplug queue up in the end
835     */
836     if ((mq = g_queue_new()) == NULL) {
837         goto err;
838     }
839     if ((devlink_hdl = di_devlink_init(NULL, 0)) == NULL) {
840         g_queue_free (mq);
841         goto err;
842     }
843     minor = DI_MINOR_NIL;
844     while ((minor = di_minor_next(node, minor)) != DI_MINOR_NIL) {
845         dev = di_minor_devt(minor);
846         if ((major != major(dev)) ||
847             (di_minor_type(minor) != DDM_MINOR) ||
848             (di_minor_spectype(minor) != S_IFBLK) ||
849             ((minor_path = di_devfs_minor_path(minor)) == NULL)) {
850             continue;
851         }
852         if ((devlink = get_devlink(devlink_hdl, NULL, minor_path)) == NU
853             di_devfs_path_free (minor_path);
854         continue;
855     }

```

```

857     slice = devinfo_volume_get_slice_name (devlink);
858     if (strlen (slice) < 2) {
859         free (devlink);
860         di_devfs_path_free (minor_path);
861         continue;
862     }

164     /* ignore p1..N - we'll use p0:N instead */
165     if ((strlen (slice) > 1) && (slice[0] == 'p') && isdigit(slice[1]
166         ((atol(&slice[1]) > 0)) {
167         free (devlink);
168         di_devfs_path_free (minor_path);
169         continue;
170     }

864     m = devinfo_storage_new_minor(minor_path, slice, devlink, dev, -
865     if (m == NULL) {
866         free (devlink);
867         di_devfs_path_free (minor_path);
868         continue;
869     }

871     /* main device is either s2/p0 or d0, the latter taking preceden
872     if ((strcmp (slice, "d0") == 0) ||
873         ((strcmp (slice, whole_disk) == 0) && (maindev == NULL))))
874         if (maindev_path != NULL) {
875             di_devfs_path_free (maindev_path);
876         }
877         maindev_path = minor_path;
878         maindev = m;
879         g_queue_push_head (mq, maindev);
880     } else {
881         di_devfs_path_free (minor_path);
882         g_queue_push_tail (mq, m);
883     }

885     free (devlink);
886 }
887 di_devlink_fini (&devlink_hdl);

889     if (maindev == NULL) {
890         /* shouldn't typically happen */
891         while (!g_queue_is_empty (mq)) {
892             devinfo_storage_free_minor (g_queue_pop_head (mq));
893         }
894         goto err;
895     }

897     /* first enqueue main storage device */
898     if (!rescan) {
899         hal_device_property_set_int (parent, "block.major", major);
900         hal_device_property_set_int (parent, "block.minor", minor(mainde
901         hal_device_property_set_string (parent, "block.device", maindev-
902         raw = dsk_to_rdisk (maindev->devlink);
903         hal_device_property_set_string (parent, "block.solaris.raw_devic
904         free (raw);
905         hal_device_property_set_bool (parent, "block.is_volume", FALSE);
906         hal_device_property_set_string (parent, "solaris.devfs_path", ma
907         devinfo_add_enqueue (parent, maindev_path, &devinfo_storage_hand
908     }

910     /* add virtual dos volumes to enable pcfs probing */
911     if (!is_cdrom) {
912         doslink_len = strlen (maindev->devlink) + sizeof ("pNN") + 1;
920         doslink_len = strlen (maindev->devlink) + sizeof (".:NNN") + 1;
913         if ((doslink = (char *)calloc (1, doslink_len)) != NULL) {

```

```

914         for (i = 1; i < 16; i++) {
915             snprintf(dospath, sizeof (dospath), "%sp%d", mai
916             snprintf(doslink, doslink_len, "%sp%d", maindev-
923             snprintf(dospath, sizeof (dospath), "%s:%d", mai
924             snprintf(doslink, doslink_len, "%s:%d", maindev-
917             m = devinfo_storage_new_minor(maindev_path, dosp
918             g_queue_push_tail (mq, m);
919         }
920         free (doslink);
921     }
922 }

924     maindev_is_d0 = (strcmp (maindev->slice, "d0") == 0);

926     /* enqueue all volumes */
927     while (!g_queue_is_empty (mq)) {
928         m = g_queue_pop_head (mq);

930         /* if main device is d0, we'll throw away s2/p0 */
931         if (maindev_is_d0 && (strcmp (m->slice, whole_disk) == 0)) {
932             devinfo_storage_free_minor (m);
933             continue;
934         }
935         /* don't do p0 on cdrom */
936         if (is_cdrom && (strcmp (m->slice, "p0") == 0)) {
937             devinfo_storage_free_minor (m);
938             continue;
939         }
940         if (rescan) {
941             /* in rescan mode, don't reprobe existing volumes */
942             /* XXX detect volume removal? */
943             volume = hal_device_store_match_key_value_string (hald_g
944                 "solaris.devfs_path", m->devpath);
945             if ((volume == NULL) || !hal_device_has_capability(volum
946                 devinfo_volume_add (parent, node, m);
947             } else {
948                 HAL_INFO(("rescan volume exists %s", m->devpath)
949             }
950         } else {
951             devinfo_volume_add (parent, node, m);
952         }
953         devinfo_storage_free_minor (m);
954     }

956     if (maindev_path != NULL) {
957         di_devfs_path_free (maindev_path);
958     }

960     return;

962 err:
963     if (maindev_path != NULL) {
964         di_devfs_path_free (maindev_path);
965     }
966     if (!rescan) {
967         devinfo_add_enqueue (parent, devfs_path, &devinfo_storage_handle
968     }
969 }

_____unchanged_portion_omitted_____

1395 static gboolean
1396 is_dos_path(char *path, int *partnum)
1397 {
1398     char *p;

1400     if ((p = strrchr (path, 'p')) == NULL) {

```

```
708     if ((p = strrchr (path, ':')) == NULL) {
1401         return (FALSE);
1402     }
1403     return ((*partnum = atoi(p + 1)) != 0);
1404 }

1406 static gboolean
1407 dos_to_dev(char *path, char **devpath, int *partnum)
1408 {
1409     char *p;

1411     if ((p = strrchr (path, 'p')) == NULL) {
719         if ((p = strrchr (path, ':')) == NULL) {
1412             return (FALSE);
1413         }
1414         if ((*partnum = atoi(p + 1)) == 0) {
1415             return (FALSE);
1416         }
1417         p[0] = '\0';
1418         *devpath = strdup(path);
1419         p[0] = 'p';
727         p[0] = ':';
1420         return (*devpath != NULL);
1421 }

unchanged_portion_omitted
```

new/usr/src/cmd/rmformat/Makefile

1

```
*****
1589 Sun May 4 18:28:35 2014
new/usr/src/cmd/rmformat/Makefile
4833 Remove volrmmount
4845 rm(u)mount don't always print mount/unmount errors
4846 HAL partition names don't match real partition names
Reviewed by: Dan McDonald <danmcd@omniti.com>
Reviewed by: Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
*****
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License, Version 1.0 only
6 # (the "License"). You may not use this file except in compliance
7 # with the License.
8 #
9 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 # or http://www.opensolaris.org/os/licensing.
11 # See the License for the specific language governing permissions
12 # and limitations under the License.
13 #
14 # When distributing Covered Code, include this CDDL HEADER in each
15 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 # If applicable, add the following below this CDDL HEADER, with the
17 # fields enclosed by brackets "[]" replaced with your own identifying
18 # information: Portions Copyright [yyyy] [name of copyright owner]
19 #
20 # CDDL HEADER END
21 #
22 #
23 # Copyright 2005 Sun Microsystems, Inc. All rights reserved.
24 # Use is subject to license terms.
25 #
26 # Copyright 2014 Andrew Stormont.
27 #
28 #endif /* ! codereview */

30 PROG= rmformat

32 OBJS=  rmf_main.o rmf_menu.o rmf_misc.o rmf_slice.o

34 include ../Makefile.cmd

36 SRCS= $(OBJS:.o=.c)

38 LDLIBS +=      -lsmedia -lvolmgt -ladm -lefi

26 CERRWARN +=  _gcc=-Wno-uninitialized

40 LINTFLAGS += -u
41 CPPFLAGS += -D_FILE_OFFSET_BITS=64

43 $(ROOTBIN)/rmformat := FILEMODE = 04555

45 .KEEP_STATE:

47 all: $(PROG)

49 $(PROG): $(OBJS)
50     $(LINK.c) -o $(PROG) $(OBJS) $(LDLIBS)
51     $(POST_PROCESS)

53 install: all $(ROOTPROG)

55 clean:
```

new/usr/src/cmd/rmformat/Makefile

2

```
56     $(RM) $(OBJS)

58 lint: lint_SRCS

60 $(POFILE) : $(SRCS)
61     $(RM) $@
62     $(COMPILE.cpp) $(SRCS) | $(XGETTEXT) $(XGETFLAGS) -
63     $(SED) -e '/^domain/d' messages.po > $@
64     $(RM) messages.po

66 sb: $(SRCS)
67     $(COMPILE.c) -xsbfast $(SRCS)

69 include ../Makefile.targ
```


new/usr/src/cmd/rmformat/rmf_menu.c

1

```
*****
30055 Sun May 4 18:28:36 2014
new/usr/src/cmd/rmformat/rmf_menu.c
4833 Remove volrmmount
4845 rm(u)mount don't always print mount/unmount errors
4846 HAL partition names don't match real partition names
Reviewed by: Dan McDonald <danmcd@omniti.com>
Reviewed by: Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 *
21 *
20 */
21 /*
22  * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
23  * Use is subject to license terms.
24  *
25  * Copyright 2014 Andrew Stormont.
26  *#endif /* ! codereview */
27  */
28
29 /*
30  * rmf_menu.c :
31  * Command line options to rmformat are processed in this file.
32  */
33
34 #include "rmformat.h"
35 #include <sys/smedia.h>
36 #include <priv_utils.h>
37
38 extern int32_t D_flag;
39 extern int32_t e_flag;
40 extern int32_t H_flag;
41 extern int32_t U_flag;
42 extern int32_t V_flag;
43 extern int32_t b_flag;
44 extern int32_t w_flag;
45 extern int32_t W_flag;
46 extern int32_t s_flag;
47 extern int32_t c_flag;
48 extern int32_t F_flag;
49 extern int32_t R_flag;
50 extern int32_t p_flag;
51 extern int32_t l_flag;
52
53 extern char *myname;
54 extern char *slice_file;
55 extern diskaddr_t repair_blk_no;
```

new/usr/src/cmd/rmformat/rmf_menu.c

2

```
56 extern int32_t quick_format;
57 extern int32_t long_format;
58 extern int32_t force_format;
59 extern int32_t rw_protect_enable;
60 extern int32_t rw_protect_disable;
61 extern int32_t wp_enable_passwd;
62 extern int32_t wp_disable_passwd;
63 extern int32_t wp_enable;
64 extern int32_t wp_disable;
65 extern int32_t verify_write;
66 extern char *dev_name;
67 extern char *label;
68 extern int total_devices_found;
69 extern int removable_found;
70 char *global_intr_msg;
71 smmedium_prop_t med_info;
72 int vol_running;
73
74 extern void check_invalid_combinations();
75 extern void check_invalid_combinations_again(int32_t);
76 extern void process_options();
77 extern void get_passwd(struct smwp_state *wp, int32_t confirm);
78 extern int32_t valid_slice_file(smedia_handle_t, int32_t, char *,
79 struct extvtoc *);
80 extern void trap_SIGINT();
81 extern void release_SIGINT();
82 extern int32_t verify(smedia_handle_t handle, int32_t fd,
83 diskaddr_t start_sector, uint32_t nblocks,
84 char *buf, int32_t flag, int32_t blocksize, int32_t no_raw_rw);
85 extern void my_perror(char *err_string);
86 extern void write_default_label(smedia_handle_t, int32_t fd);
87 extern int find_device(int defer, char *tmpstr);
88
89 void overwrite_metadata(int32_t fd, smedia_handle_t handle);
90
91 int32_t write_sunos_label(int32_t fd, int32_t media_type);
92
93 int32_t my_open(char *device_name, int32_t flags);
94
95 int32_t check_and_unmount_vold(char *device_name, int32_t flag);
96 int32_t check_and_unmount_scsi(char *device_name, int32_t flag);
97 int32_t check_and_unmount_floppy(int32_t fd, int32_t flag);
98 int32_t get_confirmation(void);
99
100 static void process_F_flag(smedia_handle_t handle, int32_t fd);
101 static void process_w_flag(smedia_handle_t handle);
102 static void process_W_flag(smedia_handle_t handle);
103 static void process_R_flag(smedia_handle_t handle);
104 void process_p_flag(smedia_handle_t handle, int32_t fd);
105 static void process_c_flag(smedia_handle_t handle);
106 static void process_V_flag(smedia_handle_t handle, int32_t fd);
107 static void process_s_flag(smedia_handle_t, int32_t fd);
108 static void process_e_flag(smedia_handle_t handle);
109 static void process_H_flag(smedia_handle_t handle, int32_t fd);
110 static void process_D_flag(smedia_handle_t handle, int32_t fd);
111 static void process_b_flag(int32_t fd);
112 static void process_l_flag(void);
113
114 void
115 process_options()
116 {
117     int32_t fd;
118     smedia_handle_t handle;
119     int32_t m_scsi_umount = 0;
```

```

119     int32_t m_flp_umount = 0;
120     int32_t v_device_umount = 0;
121     int32_t umount_required = 0;
122     int32_t removable;
123     int32_t umount_failed = 0;
124     struct dk_minfo media;

126     check_invalid_combinations();

128     if (l_flag && !dev_name) {
129         process_l_flag();
130         return;
131     }

133     if (U_flag) {
134         if (!(F_flag || H_flag || D_flag)) {
135             F_flag = 1;
136             long_format = 1;
137         }
138     }

140     if (F_flag || w_flag || W_flag || R_flag || D_flag || H_flag ||
141         V_flag || c_flag || b_flag || s_flag || e_flag) {
142         umount_required = 1;
143     }

145     fd = my_open(dev_name, O_RDONLY|O_NDELAY);
146     if (fd < 0) {
147         PERROR("Could not open device");
148         (void) close(fd);
149         exit(1);
150     }

152     if (ioctl(fd, DKIOCREMOVABLE, &removable) < 0) {
153         PERROR("DKIOCREMOVABLE ioctl failed");
154         (void) close(fd);
155         exit(1);
156     }
157     if (!removable) {
158         (void) fprintf(stderr,
159             gettext("Not a removable media device\n"));
160         (void) close(fd);
161         exit(1);
162     }

164     if (ioctl(fd, DKIOCGMEDIAINFO, &media) < 0) {
165         (void) fprintf(stderr,
166             gettext("No media in specified device\n"));
167         (void) close(fd);
168         exit(1);
169     }

171     /* Check if volume manager has mounted this */
172     if (umount_required) {
173         v_device_umount = check_and_unmount_scsi(dev_name, U_flag);
174         v_device_umount = check_and_unmount_vold(dev_name, U_flag);
175         if (v_device_umount != 1) {
176             m_flp_umount = check_and_unmount_floppy(fd, U_flag);
177             m_scsi_umount = check_and_unmount_scsi(dev_name,
178                 U_flag);
179             if (m_scsi_umount != 1) {
180                 m_flp_umount = check_and_unmount_floppy(fd,
181                     U_flag);
182                 if (m_flp_umount != 1) {
183                     umount_failed = 1;
184                 }
185             }
186         }
187     }

```

```

179     }
180     }
181     }
182     if (umount_required && U_flag && umount_failed) {
183         if (v_device_umount || m_scsi_umount || m_flp_umount) {
184             (void) fprintf(stderr,
185                 gettext("Could not unmount device.\n"));
186             (void) close(fd);
187             exit(1);
188         }
189     }

191     if (umount_required && !U_flag) {
192         if (v_device_umount || m_scsi_umount || m_flp_umount) {
193             (void) fprintf(stderr, gettext("Device mounted.\n"));
194             (void) fprintf(stderr,
195                 gettext("Requested operation can not be \
196 performed on a mounted device.\n"));
197             (void) close(fd);
198             exit(1);
199         }
200     }
201     /* register the fd with the libsmmedia */
202     handle = smedia_get_handle(fd);
203     if (handle == NULL) {
204         (void) fprintf(stderr,
205             gettext("Failed to get libsmmedia handle.\n"));
206         (void) close(fd);
207         exit(1);
208     }

210     if (smedia_get_medium_property(handle, &med_info) < 0) {
211         (void) fprintf(stderr,
212             gettext("Get medium property failed \n"));
213         (void) smedia_release_handle(handle);
214         (void) close(fd);
215         exit(1);
216     }

218     DPRINTF1("media type %x\n", med_info.sm_media_type);
219     DPRINTF1("media block size %x\n", med_info.sm_blocksize);
220     DPRINTF1("media capacity %u\n", (uint32_t)med_info.sm_capacity);
221     DPRINTF3("media cyl %d head %d sect %d\n",
222         med_info.sm_p cyl, med_info.sm_nhead, med_info.sm_nsect);
223     check_invalid_combinations_again(med_info.sm_media_type);

225     /*
226      * Special handling for pcmcia, sometimes open the file in
227      * read-write mode.
228     */

230     if (med_info.sm_media_type == SM_PCMCIA_MEM) {
231         if (F_flag || H_flag || D_flag || (V_flag && verify_write)) {
232             (void) close(fd);
233             DPRINTF("Reopening device\n");
234             fd = my_open(dev_name, O_RDWR|O_NDELAY);
235             if (fd < 0) {
236                 PERROR("Could not open device");
237                 (void) smedia_release_handle(handle);
238                 (void) close(fd);
239                 exit(1);
240             }
241         }
242     }

```

```

244     if (med_info.sm_media_type == SM_PCMCIA_ATA) {
245         if (V_flag || c_flag) {
246             (void) fprintf(stderr,
247                 gettext("Option not supported on PC ATA cards\n"));
248             (void) smedia_release_handle(handle);
249             (void) close(fd);
250             exit(1);
251         }
252         if (F_flag) {
253             /* same text as used by the format command */
254             (void) fprintf(stderr,
255                 gettext("Cannot format this drive. Please use your \
256 Manufacturer supplied formatting utility.\n"));
257             (void) smedia_release_handle(handle);
258             (void) close(fd);
259             exit(1);
260         }
261     }

263     if (F_flag)
264         process_F_flag(handle, fd);
265     if (w_flag)
266         process_w_flag(handle);
267     if (W_flag)
268         process_W_flag(handle);
269     if (R_flag)
270         process_R_flag(handle);
271     if (p_flag)
272         process_p_flag(handle, fd);
273     if (D_flag)
274         process_D_flag(handle, fd);
275     if (H_flag)
276         process_H_flag(handle, fd);
277     if (V_flag)
278         process_V_flag(handle, fd);
279     if (c_flag)
280         process_c_flag(handle);
281     if (b_flag)
282         process_b_flag(fd);
283     if (s_flag)
284         process_s_flag(handle, fd);
285     if (e_flag)
286         process_e_flag(handle);
287     if (l_flag) {
288         process_l_flag();
289     }

291     (void) smedia_release_handle(handle);
292     (void) close(fd);
293 }

295 /*
296 * This routine handles the F_flag.
297 * This options should not be used for floppy. However,
298 * if this option is used for floppy, the option will
299 * be forced to SM_FORMAT_HD and smedia_format is called.
300 * Note that smedia_format is a blocked mode format and it
301 * returns only after the complete formatting is over.
302 */

304 static void
305 process_F_flag(smedia_handle_t handle, int32_t fd)
306 {
307     uint32_t format_flag = 0;
244     uint32_t format_flag;
308     int32_t old_per = 0;

```

```

309     int32_t new_per, ret_val;

311     if (force_format) {
312         (void) fprintf(stderr,
313             gettext("Formatting disk.\n"));
314     } else {
315         (void) fprintf(stderr,
316             gettext("Formatting will erase all the data on disk.\n"));
317         if (!get_confirmation())
318             return;
319     }

321     if (quick_format)
322         format_flag = SM_FORMAT_QUICK;
323     else if (long_format)
324         format_flag = SM_FORMAT_LONG;
325     else if (force_format)
326         format_flag = SM_FORMAT_FORCE;

328     if (med_info.sm_media_type == SM_FLOPPY)
329         format_flag = SM_FORMAT_HD;

331     if ((med_info.sm_media_type != SM_FLOPPY) &&
332         (med_info.sm_media_type != SM_PCMCIA_MEM) &&
333         (med_info.sm_media_type != SM SCSI_FLOPPY)) {
334         global_intr_msg = "Interrupting format may render the \
335 medium useless";
336     } else {
337         global_intr_msg = "";
338     }
339     trap_SIGINT();

341     if (smedia_format(handle, format_flag, SM_FORMAT_IMMEDIATE) != 0) {
342         if (errno == EINVAL) {
343             (void) fprintf(stderr, gettext("Format failed.\n"));
344             (void) fprintf(stderr, gettext("The medium may not \
345 be compatible for format operation.\n"));
346             (void) fprintf(stderr, gettext("read/write surface \
347 scan may be used to get the effect of formatting.\n"));
348         } else {
349             PERROR("Format failed");
350         }
351         (void) smedia_release_handle(handle);
352         (void) close(fd);
353         exit(1);
354     }

356     /* CONSTCOND */
357     while (1) {
358         ret_val = smedia_check_format_status(handle);
359         if (ret_val == -1) {
360             if (errno != ENOTSUP) {
361                 PERROR("Format failed");
362                 (void) smedia_release_handle(handle);
363                 (void) close(fd);
364                 exit(1);
365             } else {
366                 /* Background formatting is not supported */
367                 break;
368             }
369         }
370         if (ret_val == 100) {
371             (void) printf("\n");
372             (void) fflush(stdout);
373             break;
374         }

```

```

375         new_per = (ret_val * 80)/100;
376         while (new_per >= old_per) {
377             (void) printf(".");
378             (void) fflush(stdout);
379             old_per++;
380         }
381         (void) sleep(6);
382     }
383
384     if ((med_info.sm_media_type == SM_FLOPPY) ||
385         (med_info.sm_media_type == SM_PCMCIA_MEM) ||
386         (med_info.sm_media_type == SM_SCSI_FLOPPY)) {
387         (void) write_sunos_label(fd, med_info.sm_media_type);
388     } else {
389
390         /*
391          * Iomega drives don't destroy the data in quick format.
392          * Do a best effort write to first 1024 sectors.
393          */
394
395         if (quick_format)
396             overwrite_metadata(fd, handle);
397
398         (void) write_default_label(handle, fd);
399     }
400
401     release_SIGINT();
402 }
403
404 /*
405  * List removable devices.
406  */
407 static void
408 process_l_flag()
409 {
410     int retry;
411     int removable;
412     int total_devices_found_last_time;
413     int defer = 0;
414     char *tmpstr = NULL;
415     char *tmpstr;
416
417 #define MAX_RETRIES_FOR_SCANNING 3
418
419     vol_running = volmgt_running();
420     if (vol_running)
421         defer = 1;
422     (void) printf(gettext("Looking for devices...\n"));
423     total_devices_found_last_time = 0;
424
425     /*
426      * Strip out any leading path. For example, /dev/rdisk/c3t0d0s2
427      * will result in tmpstr = c3t0d0s2. dev_name is given as input
428      * argument.
429      */
430     if (dev_name) {
431         if ((tmpstr = strrchr(dev_name, '/')) != NULL) {
432             tmpstr += sizeof(char);
433         } else {
434             tmpstr = dev_name;
435         }
436     }
437
438     for (retry = 0; retry < MAX_RETRIES_FOR_SCANNING; retry++) {
439         removable = find_device(defer, tmpstr);
440         if (removable == -1)

```

```

440         break;
441
442     /*
443      * We'll do a small sleep and retry the command if volume
444      * manager is running and no removable devices are found.
445      * This is because the device may be busy.
446      */
447     if (defer || (vol_running && (removable == 0))) {
448         if ((total_devices_found == 0) ||
449             (total_devices_found !=
450              total_devices_found_last_time)) {
451             total_devices_found_last_time =
452                 total_devices_found;
453             (void) sleep(2);
454         } else {
455             /* Do the printing this time */
456             defer = 0;
457             removable_found = 0;
458         }
459     } else
460         break;
461
462     if (removable_found == 0)
463         (void) printf(gettext("No removables found.\n"));
464 }
465 }

```

unchanged_portion_omitted

new/usr/src/cmd/rmformat/rmf_misc.c

1

```
*****
43460 Sun May  4 18:28:36 2014
new/usr/src/cmd/rmformat/rmf_misc.c
4833 Remove volrmmount
4845 rm(u)mount don't always print mount/unmount errors
4846 HAL partition names don't match real partition names
Reviewed by: Dan McDonald <danmcd@omniti.com>
Reviewed by: Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 *
21 *
20 */
22 /*
23 * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
24 * Use is subject to license terms.
25 *
26 * Copyright 2014 Andrew Stormont.
27 #endif /* ! codereview */
28 */
29 /*
30  * rmf_misc.c :
31  *      Miscellaneous routines for rmformat.
32  */
33
34 #include <sys/types.h>
35 #include <stdio.h>
36 #include <sys/mnttab.h>
37 #include <volmgt.h>
38 #include <sys/dkio.h>
39 #include <sys/fdio.h>
40 #include <sys/vtoc.h>
41 #include <sys/termios.h>
42 #include <sys/mount.h>
43 #include <ctype.h>
44 #include <signal.h>
45 #include <sys/wait.h>
46 #include <dirent.h>
47 #include <priv_utils.h>
48 #include <stdarg.h>
49 #include "rmformat.h"
50
51 /*
52  * Definitions.
53  */
54 #define SENSE_KEY(rqbuf)      (rqbuf[2] & 0xf) /* scsi error category */
```

new/usr/src/cmd/rmformat/rmf_misc.c

2

```
55 #define ASC(rqbuf)           (rqbuf[12]) /* additional sense code */
56 #define ASCQ(rqbuf)         (rqbuf[13]) /* ASC qualifier */
57
58 #define DEFAULT SCSI_TIMEOUT 60
59 #define INQUIRY_CMD          0x12
60 #define RQBUFLEN             32
61 #define CD_RW                1 /* CD_RW/CD-R */
62 #define WRITE_10_CMD         0x2A
63 #define READ_INFO_CMD       0x51
64 #define SYNC_CACHE_CMD      0x35
65 #define CLOSE_TRACK_CMD     0x5B
66 #define MODE_SENSE_10_CMD   0x5A
67 #define DEVFS_PREFIX        "/devices"
68
69 int          uscsi_error; /* used for debugging failed uscsi */
70 char         rqbuf[RQBUFLEN];
71 static uint_t total_retries;
72 static struct uscsi_cmd uscmd;
73 static char   ucdb[16];
74 uchar_t      uscsi_status, rqstatus, rqresid;
75 int          total_devices_found = 0;
76 int          removable_found = 0;
77
78 extern char  *global_intr_msg;
79 extern int   vol_running;
80 extern char  *dev_name;
81 extern int32_t m_flag;
82
83 /*
84  * ON-private functions from libvolmgt
85  */
86 int _dev_mounted(char *path);
87
88 /*
89  * Function prototypes.
90  */
91 static int my_umount(char *mountp);
92 static int my_volrmmount(char *real_name);
93 static int vol_name_to_dev_node(char *vname, char *found);
94 static int vol_lookup(char *supplied, char *found);
95 static device_t *get_device(char *user_supplied, char *node);
96 static char *get_physical_name(char *path);
97 static int lookup_device(char *supplied, char *found);
98 static void fini_device(device_t *dev);
99 static int is_cd(char *node);
100 void *my_zalloc(size_t size);
101 int err_msg(char *fmt, ...);
102 int inquiry(int fd, uchar_t *inq);
103 struct uscsi_cmd *get_uscsi_cmd(void);
104 int uscsi(int fd, struct uscsi_cmd *scmd);
105 int get_mode_page(int fd, int page_no, int pc, int buf_len,
106 uchar_t *buffer);
107 int mode_sense(int fd, uchar_t pc, int dbd, int page_len,
108 uchar_t *buffer);
109 int read_scsl16(void *addr);
110 int check_device(device_t *dev, int cond);
111 static void get_media_info(device_t *t_dev, char *sdev,
112 char *pname, char *sn);
113
114 extern void process_p_flag(smedia_handle_t handle, int32_t fd);
115
116 void my_perror(char *err_string)
117 {
118
119     int error_no;
```

```

120     if (errno == 0)
121         return;

123     error_no = errno;
124     (void) fprintf(stderr, "%s", err_string);
125     (void) fprintf(stderr, gettext(" : "));
126     errno = error_no;
127     perror("");
128 }

_____ unchanged portion omitted _____

161 int32_t
162 check_and_unmount_vold(char *device_name, int32_t flag)
163 {
164     char *real_name;
165     char *nm;
166     char tmp_path_name[PATH_MAX];
167     struct stat stat_buf;
168     int32_t ret_val = 0;
169     struct mnttab *mntp;
170     FILE *fp;
171     int nl;

173     DPRINTF1("Device name %s\n", device_name);

175     if (volmgt_running() == 0) {
176         DPRINTF("Vold not running\n");
177         return (0);
178     }
179     if ((nm = volmgt_symname(device_name)) == NULL) {
180         DPRINTF("path not managed\n");
181         real_name = media_findname(device_name);
182     } else {
183         DPRINTF("path managed as %s\n", nm);
184         real_name = media_findname(nm);
185         DPRINTF("real name %s\n", real_name);
186     }

188     if (real_name == NULL)
189         return (-1);

191     /*
192     * To find out whether the device has been mounted by
193     * volume manager...
194     *
195     * Convert the real name to a block device address.
196     * Do a partial match with the mnttab entries.
197     * Make sure the match is in the beginning to avoid if
198     * anybody puts a label similiar to volume manager path names.
199     * Then use "volrmmount -e <dev_name>" if -U flag is set.
200     */

202     nl = strlen("/vol/dev/");

204     if (strncmp(real_name, "/vol/dev/", nl) != 0)
205         return (0);
206     if (real_name[nl] == 'r') {
207         (void) snprintf(tmp_path_name, PATH_MAX, "%s%s", "/vol/dev/",
208             &real_name[nl + 1]);
209     } else {
210         (void) snprintf(tmp_path_name, PATH_MAX, "%s", real_name);
211     }
212     DPRINTF1("%s \n", tmp_path_name);
213     ret_val = stat(tmp_path_name, &stat_buf);
214     if (ret_val < 0) {
215         PERROR("Could not stat");

```

```

216         return (-1);
217     }

219     fp = fopen("/etc/mnttab", "r");

221     if (fp == NULL) {
222         PERROR("Could not open /etc/mnttab");
223         return (-1);
224     }

226     mntp = (struct mnttab *)malloc(sizeof (struct mnttab));
227     if (mntp == NULL) {
228         PERROR("malloc failed");
229         (void) fclose(fp);
230         return (-1);
231     }
232     errno = 0;
233     while (getmntent(fp, mntp) == 0) {
234         if (errno != 0) {
235             PERROR("Error with mnttab");
236             (void) fclose(fp);
237             return (-1);
238         }
239         /* Is it a probable entry? */
240         DPRINTF1(" %s \n", mntp->mnt_special);
241         if (strstr(mntp->mnt_special, tmp_path_name) !=
242             mntp->mnt_special) {
243             /* Skip to next entry */
244             continue;
245         } else {
246             DPRINTF1("Found!! %s\n", mntp->mnt_special);
247             ret_val = 1;
248             break;
249         }
250     }

252     if (ret_val == 1) {
253         if (flag) {
254             if (my_volrmmount(real_name) < 0) {
255                 ret_val = -1;
256             }
257         } else {
258             ret_val = -1;
259         }
260     }
261     (void) fclose(fp);
262     free(mntp);
263     return (ret_val);
264 }

227 /*
228 * This routine checks if a device has mounted partitions. The
229 * device name is assumed to be /dev/rdisk/cntNdNsN. So, this can
230 * be used for SCSI and PCMCIA cards.
231 * Returns
232 *     0 : if not mounted
233 *     1 : if successfully unmounted
234 *    -1 : Any error or umount failed
235 */

237 int32_t
238 check_and_unmount_scsi(char *device_name, int32_t flag)
239 {

241     struct mnttab *mntrefp;
242     struct mnttab *mntp;

```

```

243 FILE *fp;
244 char block_dev_name[PATH_MAX];
245 char tmp_name[PATH_MAX];
246 int32_t i, j;
247 int32_t unmounted = 0;

249 /*
250  * If the device name is not a character special, anyway we
251  * can not progress further
252  */

254 if (strncmp(device_name, "/dev/rdisk/c", strlen("/dev/rdisk/c")) != 0)
255     return (0);

257 (void) snprintf(block_dev_name, PATH_MAX, "/dev/%s",
258     &device_name[strlen("/dev/r")] );
259 fp = fopen("/etc/mnttab", "r");

261 if (fp == NULL) {
262     PERROR("Could not open /etc/mnttab");
263     return (-1);
264 }

266 mntrefp = (struct mnttab *)malloc(sizeof (struct mnttab));
267 if (mntrefp == NULL) {
268     PERROR("malloc failed");
269     (void) fclose(fp);
270     return (-1);
271 }

273 mntp = (struct mnttab *)malloc(sizeof (struct mnttab));
274 if (mntp == NULL) {
275     PERROR("malloc failed");
276     (void) fclose(fp);
277     free(mntrefp);
278     return (-1);
279 }

281 /* Try all the partitions */

283 (void) snprintf(tmp_name, PATH_MAX, "/dev/%s",
284     &device_name[strlen("/dev/r")] );

286 tmp_name[strlen("/dev/dsk/c0t0d0s")] = '\0';

288 errno = 0;
289 while (getmntent(fp, mntp) == 0) {
290     if (errno != 0) {
291         PERROR("Error with mnttab");
292         (void) fclose(fp);
293         return (-1);
294     }
295     /* Is it a probable entry? */
296     if (strncmp(mntp->mnt_special, tmp_name, strlen(tmp_name))) {
297         /* Skip to next entry */
298         continue;
299     }
300     for (i = 0; i < NDKMAP; i++) {
301         /* Check for ufs style mount devices */
302         (void) snprintf(block_dev_name, PATH_MAX,
303             "%s%d", tmp_name, i);

305         if (strcmp(mntp->mnt_special, block_dev_name) == 0) {
306             if (flag) {
307                 if (my_umount(mntp->mnt_mountp) < 0) {
308                     (void) fclose(fp);

```

```

309         return (-1);
310     }
311     unmounted = 1;
312 } else {
313     (void) fclose(fp);
314     return (-1);
315 }
316 /* Skip to next entry */
317 continue;
318 }

320 /* Try for :1 -> :24 for pcfs */

322 for (j = 1; j < 24; j++) {
323     (void) snprintf(block_dev_name, PATH_MAX,
324         "%s%d:%d", tmp_name, i, j);

326     if (strcmp(mntp->mnt_special,
327         block_dev_name) == 0) {
328         if (flag) {
329             if (my_umount(mntp->mnt_mountp)
330                 < 0) {
331                 (void) fclose(fp);
332                 return (-1);
333             }
334             unmounted = 1;
335         } else {
336             (void) fclose(fp);
337             return (-1);
338         }
339         /* Skip to next entry */
340         continue;
341     }
342     (void) snprintf(block_dev_name, PATH_MAX,
343         "%s%d:%c", tmp_name, i, 'b' + j);

345     if (strcmp(mntp->mnt_special,
346         block_dev_name) == 0) {
347         if (flag) {
348             if (my_umount(mntp->mnt_mountp)
349                 < 0) {
350                 (void) fclose(fp);
351                 return (-1);
352             }
353             unmounted = 1;
354         } else {
355             (void) fclose(fp);
356             return (-1);
357         }
358         /* Skip to next entry */
359         continue;
360     }
361 }

362 }

364 }

366     if (unmounted)
367         return (1);
368     return (0);
369 }

unchanged_portion_omitted

1005 static int
1006 my_volrmount(char *real_name)
1007 {

```

```

1008     int pid, rval;
1010     /* Turn on the privileges. */
1011     (void) __priv_bracket(PRIV_ON);
1013     pid = fork();
1015     /* Turn off the privileges. */
1016     (void) __priv_bracket(PRIV_OFF);
1018     /* create a child to unmount the path */
1019     if (pid < 0) {
1020         PERROR("fork failed");
1021         exit(0);
1022     }
1024     if (pid == 0) {
1025         /* the child */
1026         /* get rid of those nasty err messages */
1027         DPRINTF1("call_unmount_prog: calling %s \n",
1028             "/usr/bin/volrmmount");
1030         /* Turn on the privileges. */
1031         (void) __priv_bracket(PRIV_ON);
1032         if (execl("/usr/bin/volrmmount", "/usr/bin/volrmmount", "-e",
1033             real_name, NULL) < 0) {
1034             PERROR("volrmmount exec failed");
1035             /* Turn off the privileges */
1036             (void) __priv_bracket(PRIV_OFF);
1037             exit(-1);
1038         }
1039     } else if (waitpid(pid, &rval, 0) == pid) {
1040         if (WIFEXITED(rval)) {
1041             if (WEXITSTATUS(rval) == 0) {
1042                 DPRINTF("volrmmount: Success\n");
1043                 return (1);
1044             }
1045         }
1046     }
1047     return (-1);
1048 }

966 int
967 find_device(int defer, char *tmpstr)
968 {
969     DIR *dir;
970     struct dirent *dirent;
971     char sdev[PATH_MAX], dev[PATH_MAX], *pname;
972     device_t *t_dev;
973     int removable = 0;
974     int device_type = 0;
975     int hotpluggable = 0;
976     struct dk_minfo mediainfo;
977     static int found = 0;
979     dir = opendir("/dev/rdisk");
980     if (dir == NULL)
981         return (-1);
983     total_devices_found = 0;
984     while ((dirent = readdir(dir)) != NULL) {
985         if (dirent->d_name[0] == '.') {
986             continue;
987         }
988         (void) snprintf(sdev, PATH_MAX, "/dev/rdisk/%s",
989             dirent->d_name);

```

```

990 #ifdef sparc
991     if (!strstr(sdev, "s2")) {
992         continue;
993     }
994 #else /* x86 */
995     if (vol_running) {
996         if (!(strstr(sdev, "s2") || strstr(sdev, "p0"))) {
997             continue;
998         }
999     } else {
1000         if (!strstr(sdev, "p0")) {
1001             continue;
1002         }
1003     }
1004 #endif
1005     if (!lookup_device(sdev, dev)) {
1006         continue;
1007     }
1008     if ((t_dev = get_device(NULL, dev)) == NULL) {
1009         continue;
1010     }
1011     total_devices_found++;
1013     if (!(defer) && !found) {
1014         char *sn, *tmpbuf = NULL;
1015         char *sn, *tmpbuf;
1016         /*
1017          * dev_name is an optional command line input.
1018          */
1019         if (dev_name) {
1020             if (strstr(dirent->d_name, tmpstr)) {
1021                 found = 1;
1022             } else if (!vol_running) {
1023                 continue;
1024             }
1025         }
1026         /*
1027          * volmgt_symname() returns NULL if the device
1028          * is not managed by volmgt.
1029          */
1030         sn = volmgt_symname(sdev);
1031         if (vol_running && (sn != NULL)) {
1032             if (strstr(sn, "dev") == NULL) {
1033                 tmpbuf = (char *)my_zalloc(PATH_MAX);
1034                 (void) strcpy(tmpbuf,
1035                     "/vol/dev/aliases/");
1036                 (void) strcat(tmpbuf, sn);
1037                 free(sn);
1038                 sn = tmpbuf;
1039             }
1040             if (dev_name && !found) {
1041                 if (!strstr(tmpbuf, tmpstr)) {
1042                     continue;
1043                 } else {
1044                     found = 1;
1045                 }
1046             }
1047         }
1049         /*
1050          * Get device type information for CD/DVD devices.
1051          */
1052         if (is_cd(dev)) {
1053             if (check_device(t_dev,
1054                 CHECK_DEVICE_IS_DVD_WRITABLE)) {

```



```

1055         device_type = DK_DVDR;
1056     } else if (check_device(t_dev,
1057         CHECK_DEVICE_IS_DVD_READABLE)) {
1058         device_type = DK_DVDROM;
1059     } else if (check_device(t_dev,
1060         CHECK_DEVICE_IS_CD_WRITABLE)) {
1061         device_type = DK_CDR;
1062     } else {
1063         device_type = DK_CDROM;
1064     }
1065 } else {
1066     device_type = ioctl(t_dev->d_fd,
1067         DKIOCGMEDIAINFO, &mediainfo);
1068     if (device_type < 0)
1069         device_type = 0;
1070     else
1071         device_type = mediainfo.dki_media_type;
1072 }

1074 if (!ioctl(t_dev->d_fd, DKIOCREMOVABLE, &removable) &&
1075     !ioctl(t_dev->d_fd, DKIOCHOTPLUGGABLE,
1076     &hotpluggable)) {
1077     if (removable || hotpluggable) {
1078         removable_found++;
1079         pname = get_physical_name(sdev);
1080         if (sn) {
1081             (void) printf(" %4d. "
1082                 "Volmgt Node: %s\n",
1083                 removable_found, sn);
1084             (void) printf(" "
1085                 "Logical Node: %s\n", sdev);
1086             (void) printf(" "
1087                 "Physical Node: %s\n",
1088                 pname);
1089         } else {
1090             (void) printf(" %4d. "
1091                 "Logical Node: %s\n",
1092                 removable_found, sdev);
1093             (void) printf(" "
1094                 "Physical Node: %s\n",
1095                 pname);
1096         }
1097         (void) printf("          Connected "
1098             "Device: %-8.8s %-16.16s "
1099             "%-4.4s\n",
1100             &t_dev->d_inq[8],
1101             &t_dev->d_inq[16],
1102             &t_dev->d_inq[32]);
1103         (void) printf("          Device "
1104             "Type: ");
1105     } else
1106         continue;
1107 } else
1108     continue;

1110 switch (device_type) {
1111     case DK_CDROM:
1112         (void) printf("CD Reader\n");
1113         break;
1114     case DK_CDR:
1115     case DK_CDRW:
1116         (void) printf("CD Reader/Writer\n");
1117         break;
1118     case DK_DVDROM:
1119         (void) printf("DVD Reader\n");
1120         break;

```

```

1121     case DK_DVDR:
1122     case DK_DVDRAM:
1123         (void) printf("DVD Reader/Writer\n");
1124         break;
1125     case DK_FIXED_DISK:
1126         if (strstr((const char *)
1127             &t_dev->d_inq[16], "FD") ||
1128             strstr((const char *)
1129                 &t_dev->d_inq[16], "LS-120"))
1130             (void) printf("Floppy "
1131                 "drive\n");
1132         else
1133             (void) printf("Removable\n");
1134         break;
1135     case DK_FLOPPY:
1136         (void) printf("Floppy drive\n");
1137         break;
1138     case DK_ZIP:
1139         (void) printf("Zip drive\n");
1140         break;
1141     case DK_JAZ:
1142         (void) printf("Jaz drive\n");
1143         break;
1144     default:
1145         (void) printf("<Unknown>\n");
1146         DPRINTF1("\t %d\n", device_type);
1147         break;
1148     }
1149     get_media_info(t_dev, sdev, pname, sn);
1150 }
1151     fini_device(t_dev);
1152 }

1154     (void) closedir(dir);
1155     return (removable_found);
1156 }

```

unchanged_portion_omitted

```

*****
39066 Sun May 4 18:28:36 2014
new/usr/src/cmd/rmformat/rmf_slice.c
4833 Remove volrmmount
4845 rm(u)mount don't always print mount/unmount errors
4846 HAL partition names don't match real partition names
Reviewed by: Dan McDonald <danmcd@omniti.com>
Reviewed by: Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 *
21 *
22 * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 *
25 * Copyright 2014 Andrew Stormont.
26 #endif /* ! codereview */
27 */
28
29 /*
30  * rmf_slice.c :
31  * This file contains the functions for parsing a slice file
32  * for rmformat.
33  */
34
35 #include <sys/types.h>
36 #include <ctype.h>
37 #include <sys/vtoc.h>
38 #include <stdlib.h>
39 #include <unistd.h>
40 #include <string.h>
41 #include <fcntl.h>
42 #include <errno.h>
43 #include <memory.h>
44 #include <dirent.h>
45 #include <sys/fcntl.h>
46 #include <sys/param.h>
47 #include <sys/stat.h>
48 #include <stdio.h>
49 #include <sys/dkio.h>
50 #include <priv_utils.h>
51 #include "rmformat.h"
52
53 extern void my_perror(char *err_string);
54
55 static int32_t last_token_type = 0;

```

```

56 #define spc() (last_token_type)

59 /*
60  * This global is used to store the current line # in the
61  * data file. It must be global because the I/O routines
62  * are allowed to side effect it to keep track of backslashed
63  * newlines.
64  */

66 static int32_t data_lineno; /* current line # in data file */

68 #define CHG_MODE_UNDEFINED (-1) /* undefined value */
69 #define CHG_MODE_SET 0 /* set bits by or'ing */
70 #define CHG_MODE_CLR 1 /* clr bits by and'ing */
71 #define CHG_MODE_ABS 2 /* set absolute value */

74 #define TOKEN_SIZE 36 /* max length of a token */
75 typedef char TOKEN[TOKEN_SIZE+1]; /* token type */
76 #define DATA_INPUT 0 /* 2 modes of input */
77 #define CMD_INPUT 1
78 #define WILD_STRING "$" /* wildcard character */
79 #define COMMENT_CHAR '#' /* comment character */

81 /*
82  * List of strings with arbitrary matching values
83  */
84 typedef struct slist {
85     char *str;
86     char *help;
87     int32_t value;
88 } slist_t;

90 static slist_t ptag_choices[] = {
91     { "unassigned", "", V_UNASSIGNED },
92     { "boot", "", V_BOOT },
93     { "root", "", V_ROOT },
94     { "swap", "", V_SWAP },
95     { "usr", "", V_USR },
96     { "backup", "", V_BACKUP },
97     { "stand", "", V_STAND },
98     { "var", "", V_VAR },
99     { "home", "", V_HOME },
100    { "alternates", "", V_ALTSCTR },
101    { NULL }
102 };

105 /*
106  * Choices for the p_flag vtoc field
107  */
108 static slist_t pflag_choices[] = {
109     { "wm", "read-write, mountable", 0 },
110     { "wu", "read-write, unmountable", V_UNMNT },
111     { "rm", "read-only, mountable", V_RDONLY },
112     { "ru", "read-only, unmountable", V_RDONLY|V_UNMNT },
113     { NULL }
114 };

116 /*
117  * The definitions are the token types that the data file parser recognizes.
118  */
119 #define SUP_EOF -1 /* eof token */
120 #define SUP_STRING 0 /* string token */
121 #define SUP_EQ 1 /* equals token */

```

```

122 #define SUP_COMMA      2      /* comma token */
123 #define SUP_COLON     3      /* colon token */
124 #define SUP_EOL       4      /* newline token */
125 #define SUP_OR        5      /* vertical bar */
126 #define SUP_AND       6      /* ampersand */
127 #define SUP_TILDE     7      /* tilde */

130 /*
131  * Prototypes for ANSI C compilers
132  */
133 static int32_t sup_prxfile(char *file_name, struct extvtoc *vt);
134 static int32_t sup_setpart(struct extvtoc *vt);
135 static void sup_pushchar(int32_t c);
136 static void clean_token(char *cleantoken, char *token);
137 static void clean_token(char *cleantoken, char *token);
138 static int32_t sup_inputchar();
139 static int32_t sup_gettoken(char *buf);
140 static int32_t sup_get_token(char *buf);
141 static int32_t find_value(slist_t *slist, char *str, int32_t *value);
142 static int32_t check_vtoc_sanity(smmedia_handle_t, int32_t fd,
143     struct extvtoc *vt);
144 static uint64_t str2sector(char *str);
145 static int32_t strcnt(char *s1, char *s2);
146 static int32_t get_fdisk(smmedia_handle_t, int32_t fd, int32_t offset,
147     struct fdisk_info *fdisk);
148 static void erase(smmedia_handle_t handle, diskaddr_t offset, diskaddr_t size);

150 extern char *myname;
151 extern uint64_t my_atoll(char *ptr);
152 extern smmedium_prop_t med_info;

154 static FILE *data_file;

156 static int32_t
157 sup_prxfile(char *file_name, struct extvtoc *vt)
158 {
159     int32_t status, ret_val;
160     TOKEN token;
161     TOKEN cleaned;

163     /*
164      * Open the data file. Return 0 if unable to do so.
165      */
166     data_file = fopen(file_name, "r");
167     if (data_file == NULL) {
168         PERROR("Open failed");
169         return (-1);
170     }
171     /*
172      * Step through the data file a meta-line at a time. There are
173      * typically several backslashed newlines in each meta-line,
174      * so data_lineno will be getting side effected along the way.
175      */
176     data_lineno = 1;
177     for (;;) {

179         /*
180          * Get the keyword.
181          */
182         status = sup_gettoken(token);
183         /*
184          * If we hit the end of the data file, we're done.
185          */
186         if (status == SUP_EOF)
187             break;

```

```

188     /*
189      * If the line starts with some key character, it's an error.
190      */
191     if (status != SUP_STRING) {
192         (void) fprintf(stderr,
193             gettext("Expecting keyword, found '%s'"),
194             token);
195         (void) fprintf(stderr,
196             gettext("Line no %d\n"), data_lineno);
197         continue;
198     }
199     /*
200      * Clean up the token and see which keyword it is. Call
201      * the appropriate routine to process the rest of the line.
202      */
203     clean_token(cleaned, token);
204     if (strcmp(cleaned, "slices") == 0) {
205         ret_val = sup_setpart(vt);
206         (void) fclose(data_file);
207         return (ret_val);
208     } else {
209         (void) fprintf(stderr, gettext("Unknown keyword '%s'",
210             cleaned));
211         (void) fprintf(stderr,
212             gettext("Line no %d\n"), data_lineno);
213         (void) fclose(data_file);
214         return (-1);
215     }
216 }
217 /*
218  * Close the data file.
219  */
220 (void) fclose(data_file);

222 (void) fprintf(stderr,
223     gettext("Unexpected end of file (line no %d)\n"), data_lineno);
224 return (-1);
225 }

227 static int32_t
228 sup_gettoken(char *buf)
229 {
230     /*
231      * Skip end of lines and blank lines.
232      */
233     while ((last_token_type = sup_get_token(buf)) == SUP_EOL)
234         ;
235     return (last_token_type);
236 }

238 static int32_t
239 sup_get_token(char *buf)
240 {
241     char *ptr = buf;
242     int32_t c, quoted = 0;

244     /*
245      * Was an end of file detected last try?
246      */

248     if (feof(data_file)) {
249         return (SUP_EOF);
250     }

252     /*
253      * Zero out the returned token buffer

```

```

254     */
255     bzero(buf, TOKEN_SIZE + 1);
256
257     /*
258     * Strip off leading white-space.
259     */
260     while (isspace(c = sup_inputchar()))
261         ;
262
263     /*
264     * Only white spaces and then end of file?
265     */
266
267     if (feof(data_file)) {
268         return (SUP_EOF);
269     }
270
271     /*
272     * Read in characters until we hit unquoted white-space.
273     */
274     for (; !isspace(c) || quoted; c = sup_inputchar()) {
275
276         /*
277         * If we hit eof, check if we have anything in buffer.
278         * If we have, return STRING, next time we will return EOF
279         * else, return EOF here...should not happen.
280         */
281         if (feof(data_file)) {
282             if (ptr - buf > 0) {
283                 return (SUP_STRING);
284             } else {
285                 return (SUP_EOF);
286             }
287         }
288
289         /*
290         * If we hit a double quote, change the state of quoting.
291         */
292         if (c == '"') {
293             quoted = !quoted;
294             continue;
295         }
296
297         /*
298         * If we hit a newline, that delimits a token.
299         */
300         if (c == '\n')
301             break;
302
303         /*
304         * If we hit any nonquoted special delimiters, that delimits
305         * a token.
306         */
307         if (!quoted && (c == '=' || c == ',' || c == ':' ||
308             c == '#' || c == '|' || c == '&' || c == '~'))
309             break;
310
311         /*
312         * Store the character if there's room left.
313         */
314         if (ptr - buf < TOKEN_SIZE)
315             *ptr++ = (char)c;
316     }
317     /*
318     * If we stored characters in the buffer, then we inputted a string.
319     * Push the delimiter back into the pipe and return the string.
320     */
321     if (ptr - buf > 0) {

```

```

320         sup_pushchar(c);
321         return (SUP_STRING);
322     }
323     /*
324     * We didn't input a string, so we must have inputted a known delimiter.
325     * store the delimiter in the buffer, so it will get returned.
326     */
327     buf[0] = c;
328     /*
329     * Switch on the delimiter. Return the appropriate value for each one.
330     */
331     switch (c) {
332     case '=':
333         return (SUP_EQ);
334     case ':':
335         return (SUP_COLON);
336     case ',':
337         return (SUP_COMMA);
338     case '\n':
339         return (SUP_EOL);
340     case '|':
341         return (SUP_OR);
342     case '&':
343         return (SUP_AND);
344     case '~':
345         return (SUP_TILDE);
346     case '#':
347         /*
348         * For comments, we flush out the rest of the line and return
349         * an eol.
350         */
351         while ((c = sup_inputchar()) != '\n' && !feof(data_file))
352             ;
353         if (feof(data_file))
354             return (SUP_EOF);
355         else
356             return (SUP_EOL);
357     }
358     /*
359     * Shouldn't ever get here.
360     */
361     default:
362         return (SUP_STRING);
363 }
364 static int32_t
365 sup_inputchar()
366 {
367     int32_t c;
368
369     /*
370     * Input the character.
371     */
372     c = getc(data_file);
373     /*
374     * If it's not a backslash, return it.
375     */
376
377     /*
378     * It was a backslash. Get the next character.
379     */
380
381     if (c == '\\')
382         c = getc(data_file);
383
384     /*
385     * If it was a newline, update the line counter and get the next

```

```

386     * character.
387     */
388     if (c == '\n') {
389         data_lineno++;
390     }
391     /*
392     * Return the character.
393     */
394     return (c);
395 }

397 static void
398 sup_pushchar(int32_t c)
399 {
401     (void) ungetc(c, data_file);
402     if (c == '\n')
403         data_lineno--;
404 }

406 static void
407 clean_token(char *cleantoken, char *token)
408 {
409     char *ptr;
411     /*
412     * Strip off leading white-space.
413     */
414     for (ptr = token; isspace(*ptr) && (ptr <=
415         (token + strlen(token) - 1)); ptr++)
416         ;
418     /*
419     * Copy it into the clean buffer.
420     */
421     (void) strcpy(cleantoken, ptr);
422     /*
423     * Strip off trailing white-space.
424     */
425     for (ptr = cleantoken + strlen(cleantoken) - 1;
426         isspace(*ptr) && (ptr >= cleantoken); ptr--) {
427         *ptr = '\0';
428     }
429 }

431 static int32_t
432 sup_setpart(struct extvtoc *vt)
433 {
434     TOKEN token, cleaned, ident;
435     int32_t i, index, status;
436     uint64_t val1, val2;
437     ushort_t vtag = 0xFFFF;
438     ushort_t vflag = 0xFFFF;
440     /*
441     * Pull in some grammar.
442     */
444     status = sup_gettoken(token);
446     if (status != SUP_COLON) {
447         (void) fprintf(stderr,
448             gettext("Expecting ':', found '%s'", token);
449             (void) fprintf(stderr,
450                 gettext("Line no %d\n", data_lineno);
451                 return (-1);

```

```

452     }
454     for (;;) {
455         status = sup_gettoken(token);
456         if (status != SUP_STRING) {
457             (void) fprintf(stderr,
458                 gettext("Expecting string, found '%s'", token);
459                 (void) fprintf(stderr,
460                     gettext("Line no %d\n", data_lineno);
461                     return (-1);
462             }
463         clean_token(ident, token);
464         /*
465         * Here's the index of the partition we're dealing with
466         */
467         index = (int32_t)my_atoll(ident);
468         if ((index < 0) || (index >= NDKMAP)) {
469             (void) fprintf(stderr,
470                 gettext("Unknown partition %d", index);
471                 (void) fprintf(stderr,
472                     gettext("Line no %d\n", data_lineno);
473                     return (-1);
474             }
475         /*
476         * Check for floppy and PCMCIA_MEM cards.
477         * for floppy, the partition no. can be 0 1 2.
478         * for PCMCIA, the partition no. can be 2
479         */
480         if (med_info.sm_media_type == SM_FLOPPY) {
481             if ((index < 0) || (index > 2)) {
482                 (void) fprintf(stderr, gettext(
483                     "Floppy can have partitions 0 1 and 2\n"));
484                 return (-1);
485             }
486         }
487         if (med_info.sm_media_type == SM_PCPCIA_MEM) {
488             if (index != 2) {
489                 (void) fprintf(stderr, gettext(
490                     "PCMCIA Memory cards can have partition 2 only.\n"));
491                 return (-1);
492             }
493         }
495         DPRINTF1("\n Partition %d: ", index);
497         status = sup_gettoken(token);
498         if (status != SUP_EQUAL) {
499             (void) fprintf(stderr,
500                 gettext("Expecting '=', found '%s'", token);
501                 (void) fprintf(stderr,
502                     gettext("Line no %d\n", data_lineno);
503                     return (-1);
504             }
505         }
508         status = sup_gettoken(token);
509         /*
510         * If we hit a key character, it's an error.
511         */
512         if (status != SUP_STRING) {
513             (void) fprintf(stderr,
514                 gettext("Expecting value, found '%s'", token);
515                 (void) fprintf(stderr,
516                     gettext("Line no %d\n", data_lineno);
517                     return (-1);

```

```

518     }
519     clean_token(cleaned, token);
520     /*
521     * <tag> may be one of: boot, root, swap, etc.
522     * <flag> consists of two characters:
523     *   W (writable) or R (read-only)
524     *   M (mountable) or U (unmountable)
525     *
526     * Start with the defaults assigned above:
527     */

529     /*
530     * All other attributes have a pair of numeric values.
531     * Convert the first value to a number. This value
532     * is the starting cylinder number of the partition.
533     */

535     /* Check for valid partition, e.g. > 8 or 16 */
536     vall = str2sector(cleaned);
537     if (vall == -1) {
538         (void) fprintf(stderr,
539             gettext("Invalid partition beggining %s \n"),
540             cleaned);
541         (void) fprintf(stderr,
542             gettext("Line no %d\n"), data_lineno);
543     }

545     DPRINTF1(" begins %s", cleaned);
546     /*
547     * Pull in some grammar.
548     */
549     status = sup_gettoken(token);
550     if (status != SUP_COMMA) {
551         (void) fprintf(stderr,
552             gettext("Expecting ', ', found '%s'", token);
553         (void) fprintf(stderr,
554             gettext("Line no %d\n"), data_lineno);
555         return (-1);
556     }
557     /*
558     * Pull in the second value.
559     */
560     status = sup_gettoken(token);
561     if (status != SUP_STRING) {
562         (void) fprintf(stderr,
563             gettext("Expecting value, found '%s'", token);
564         (void) fprintf(stderr,
565             gettext("Line no %d\n"), data_lineno);
566         return (-1);
567     }
568     clean_token(cleaned, token);

570     val2 = str2sector(cleaned);
571     if (val2 == -1) {
572         (void) fprintf(stderr,
573             gettext("Invalid partition size %s \n"),
574             cleaned);
575         (void) fprintf(stderr,
576             gettext("Line no %d\n"), data_lineno);
577     }
578     DPRINTF1(" ends %s ", cleaned);

580     /*
581     * Pull in some grammar.
582     */
583     status = sup_gettoken(token);

```

```

585         if (status == SUP_COMMA) {
586             /* tags and flags */
587             status = sup_gettoken(token);
588             if (status != SUP_STRING) {
589                 (void) fprintf(stderr,
590                     gettext("Expecting value, found '%s'",
591                         token);
592                 (void) fprintf(stderr,
593                     gettext("Line no %d\n"), data_lineno);
594                 return (-1);
595             }
596             clean_token(cleaned, token);
597             if (find_value(pflag_choices, cleaned, &i) == 1) {
598                 /*
599                 * Found valid tag. Use it and advance parser
600                 */
601                 DPRINTF1(" flag = %s", cleaned);
602                 vtoc_flag = (ushort_t)i;
603                 status = sup_gettoken(token);
604             } else if (find_value(ptag_choices, cleaned, &i) == 1) {
605                 DPRINTF1(" tag = %s", cleaned);
606                 vtoc_tag = (ushort_t)i;
607                 status = sup_gettoken(token);
608                 if (status == SUP_COMMA) {
609                     (void) fprintf(stderr,
610                         gettext("Expecting : got %s\n",
611                             token);
612                     (void) fprintf(stderr,
613                         gettext("Line no %d\n"),
614                             data_lineno);
615                     return (-1);
616                 }
617             } else {
618                 (void) fprintf(stderr,
619                     gettext("Invalid flag or tag\n"));
620                 (void) fprintf(stderr,
621                     gettext("Line no %d\n"), data_lineno);
622                 return (-1);
623             }
624         }

626         if (status == SUP_COMMA) {
627             /* Can be tag only */

629             status = sup_gettoken(token);
630             if (status != SUP_STRING) {
631                 (void) fprintf(stderr,
632                     gettext("Expecting value"
633                         ", found '%s'",
634                             token);
635                 (void) fprintf(stderr,
636                     gettext("Line no %d\n"),
637                             data_lineno);
638                 return (-1);
639             }

641             clean_token(cleaned, token);
642             if (find_value(ptag_choices,
643                 cleaned, &i) == 1) {
644                 DPRINTF1(" tag = %s", cleaned);
645                 vtoc_tag = (ushort_t)i;
646             }
647             status = sup_gettoken(token);
648         }
649     }

```

```

651     /*
652      * Fill in the appropriate map entry with the values.
653      */
654     vt->v_part[index].p_start = val1;
655     vt->v_part[index].p_size = val2;
656     if (vtoc_tag != 0xFFFF) {
657         vt->v_part[index].p_tag = vtoc_tag;
658         vtoc_tag = 0xFFFF;
659     }
660     if (vtoc_flag != 0xFFFF) {
661         vt->v_part[index].p_flag = vtoc_flag;
662         vtoc_flag = 0xFFFF;
663     }
664     if (status == SUP_EOF) {
665         DPRINTF("\nEnd of file\n");
666         break;
667     }
668     if (status != SUP_COLON) {
669         (void) fprintf(stderr,
670             gettext("Expecting ':', found '%s'", token));
671         (void) fprintf(stderr,
672             gettext("Line no %d\n"), data_lineno);
673         return (-1);
674     }
675 }
676 }
677 return (0);
678 }

```

```

680 static int32_t
681 find_value(slist_t *slist, char *match_str, int32_t *match_value)
682 {
683     int32_t i;
684     int32_t nmatches;
685     int32_t length;
686     int32_t match_length;

```

```

688     nmatches = 0;
689     length = 0;

```

```

691     match_length = strlen(match_str);

```

```

693     for (; slist->str != NULL; slist++) {
694         /*
695          * See how many characters of the token match
696          */
697         i = strcnt(match_str, slist->str);
698         /*
699          * If it's not the whole token, then it's not a match.
700          */
701         if (i < match_length) {
702             continue;
703         }
704         /*
705          * If it ties with another input, remember that.
706          */
707         if (i == length)
708             nmatches++;
709         /*
710          * If it matches the most so far, record that.
711          */
712         if (i > length) {
713             *match_value = slist->value;
714             nmatches = 1;
715             length = i;

```

```

716     }
717 }

```

```

719     return (nmatches);
720 }

```

```

722 static int32_t
723 strcnt(char *s1, char *s2)
724 {
725     int32_t i = 0;

```

```

727     while ((*s1 != '\0') && (*s1++ == *s2++))
728         i++;
729     return (i);
730 }

```

```

732 static uint64_t
733 str2sector(char *str)
734 {
735     int32_t mul_factor = 1;
736     char *s1, *s2, *base;
737     uint64_t num_sectors;
738     uint64_t size;

```

```

740     base = s2 = (char *)malloc(strlen(str) + 1);
741     if (s2 == NULL) {
742         PERROR("Malloc failed");
743         return (-1);
744     }
745     *s2 = '\0';

```

```

749     s1 = str;
750     while (*s1) {
751         if ((*s1 != 'x') && ((*s1 < 'A') || (*s1 > 'F')) &&
752             ((*s1 < 'a') || (*s1 > 'f')) && ((*s1 < '0') ||
753             (*s1 > '9'))) {
754             if (*s1 == 'G') {
755                 mul_factor = 1024*1024*1024;
756                 s1++;
757             } else if (*s1 == 'M') {
758                 mul_factor = 1024*1024;
759                 s1++;
760             } else if (*s1 == 'K') {
761                 mul_factor = 1024;
762                 s1++;
763             }
764             if ((*s1 != 'B') || (*(++s1) != NULL)) {
765                 (void) fprintf(stderr,
766                     gettext("Extra chars at the end\n"));
767                 free(base);
768                 return (-1);
769             }
770             break;
771         } else {
772             *s2++ = *s1++;
773             *s2 = '\0';
774         }
775     }
776     *s2 = NULL;

```

```

778     size = my_atoll(base);
779     if ((!mul_factor) || (size == -1)) {
780         free(base);
781         return (-1);

```

```

782     }
783     num_sectors = size * (uint64_t)mul_factor / 512;

785     free(base);
786     return (num_sectors);
787 }

790 int32_t
791 valid_slice_file(smedia_handle_t handle, int32_t fd, char *file_name,
792                 struct extvtoc *vt)
793 {
794     struct stat status;
795     int32_t ret_val;
796     if (stat(file_name, &status)) {
797         PERROR(file_name);
798         return (-1);
799     }
800     (void) memset(vt, 0, sizeof (*vt));
801     /* Set default tag and flag */
802 #ifdef sparc
803     vt->v_part[0].p_tag = V_ROOT;
804     vt->v_part[1].p_tag = V_SWAP;
805     vt->v_part[2].p_tag = V_BACKUP;
806     vt->v_part[6].p_tag = V_USR;

808     vt->v_part[1].p_flag = V_UNMNT; /* Unmountable */
809     vt->v_part[2].p_flag = V_UNMNT; /* Unmountable */
810 #endif

812     ret_val = sup_prxfile(file_name, vt);
813     if (ret_val < 0)
814         return (-1);

816 #ifdef DEBUG
817 {
818     int32_t i;
819     for (i = 0; i < 8; i++) {
820         DPRINTF1("\npart %d\n", i);
821         DPRINTF1("\t start %llu", vt->v_part[i].p_start);
822         DPRINTF1("\t size %llu ", vt->v_part[i].p_size);
823         DPRINTF1("\t tag %d", vt->v_part[i].p_tag);
824         DPRINTF1("\t flag %d", vt->v_part[i].p_flag);
825     }
826 }
827 #endif /* DEBUG */
828     if (check_vtoc_sanity(handle, fd, vt) < 0) {
829         return (-1);
830     }
831 #ifdef DEBUG
832 {
833     int32_t i;
834     for (i = 0; i < 8; i++) {
835         DPRINTF1("\npart %d\n", i);
836         DPRINTF1("\t start %llu", vt->v_part[i].p_start);
837         DPRINTF1("\t size %llu ", vt->v_part[i].p_size);
838         DPRINTF1("\t tag %d", vt->v_part[i].p_tag);
839         DPRINTF1("\t flag %d", vt->v_part[i].p_flag);
840     }
841 }
842 #endif /* DEBUG */
843     return (0);
844 }

846 #define SWAP(a, b)    {diskaddr_t tmp; tmp = (a); (a) = (b); (b) = tmp;}

```

```

848 /*
849  * On x86 Solaris, the partitioning is done in two levels, fdisk and Solaris
850  * VTOC. Where as, on sparc solaris, it is only VTOC. On floppy and PCMCIA
851  * also it is assumed to be only VTOC, no fdisk.
852  *
853  * On sparc, the back up slice can cover the whole medium. But on x86
854  * (SCSI/ATAPI disks), the backup slice can cover the solaris partition
855  * in fdisk table.
856  *     Following table describes how is it handled
857  * SPARC:
858  *     SCSI/ATAPI, floppy, pcmcia : don't check for fdisk.
859  *     DKIOCGGGEOM is sufficient.
860  * x86 : floppy, pcmcia : Don't check for fdisk. DKIOCGGGEOM is sufficient.
861  *     SCSI/ATAPI : Check for fdisk.
862  *     if not present, assume that the solaris
863  *     partition covers 100% of the medium
864  *     (minus one cylinder).
865  *
866  *     if present :
867  *     check for active solaris partition.
868  *     if not found, take the first solaris
869  *     partition.
870  *     If there are no solaris partitions, its an error, stop.
871  */

873 static int32_t
874 check_vtoc_sanity(smedia_handle_t handle, int32_t fd, struct extvtoc *vt)
875 {
876     int32_t i, j;
877     struct dk_geom dkg;
878     int32_t num_backup = 0;
879     diskaddr_t backup_size = 0;
880     struct part_struct {
881         diskaddr_t start;
882         diskaddr_t end;
883         int32_t num;
884     } part[NDKMAP];
885     diskaddr_t min_val;
886     int32_t min_slice, num_slices;
887     diskaddr_t media_size;
888     uint32_t cyl_size = 0;
889     uint32_t cyl_size;
890     int sparc_style = 0; /* sparc_style handling ? */
891     struct fdisk_info fdisk;
892     int sol_part;
893     int total_parts = 0;

895 #ifdef sparc
896     sparc_style = 1;
897 #endif /* sparc */

899     if ((med_info.sm_media_type == SM_FLOPPY) ||
900         (med_info.sm_media_type == SM_PCMCIA_MEM) ||
901         (med_info.sm_media_type == SM_PCMCIA_ATA) ||
902         (med_info.sm_media_type == SM SCSI_FLOPPY)) {
903         sparc_style = 1;
904     }

906     if (sparc_style) {
907         DPRINTF("sparc style true\n");
908         if (ioctl(fd, DKIOCGGGEOM, &dkg) < 0) {
909             PERROR("DKIOCGGGEOM Failed");
910             return (-1);
911         }
912         media_size = (diskaddr_t)dkg.dkg_ncyl * dkg.dkg_nhead *

```



```

913         dkg.dkg_nsect;
914         cyl_size = dkg.dkg_nhead * dkg.dkg_nsect;
915     }

917     if (!sparc_style) {
918         /*
919          * Try to get the fdisk information if available.
920          */
921         if (get_fdisk(handle, fd, 0, &fdisk) >= 0) {
922             /* fdisk table on disk */
923             sol_part = 0xFF;
924             for (i = 0; i < FD_NUMPART; i++) {
925                 if (fdisk.part[i].systid == SUNIXOS ||
926                     fdisk.part[i].systid == SUNIXOS2) {
927                     if (sol_part == 0xFF)
928                         sol_part = i;
929                     total_parts++;
930                     if (fdisk.part[i].bootid == ACTIVE)
931                         sol_part = i;
932                 }
933             }
934             if (sol_part == 0xFF) {
935                 /* No Solaris partition */

937                 (void) fprintf(stderr, gettext("No FDISK \
938 Solaris partition found!\n"));
939                 return (-1);
940             }
941             if (total_parts > 1)
942                 (void) fprintf(stderr, gettext("Multiple FDISK \
943 Solaris partitions found.\n"));
944             media_size = (diskaddr_t)fdisk.part[sol_part].numsect;

946             DPRINTF1("sol_part %d\n", sol_part);
947             DPRINTF1("media_size %llu\n", media_size);
948         } else {
949             DPRINTF("Didn't get fdisk\n");
950             /*
951              * No fdisk partition available. Assume a 100% Solaris.
952              * partition.
953              * Try getting disk geometry.
954              */
955             if (ioctl(fd, DKIOCGGEO, &dkg) < 0)
956                 if (ioctl(fd, DKIOCG_PHYGEO, &dkg) < 0) {
957                     DPRINTF("DKIOCG_PHYGEO ioctl failed");
958                     return (-1);
959                 }
960             /* On x86 platform 1 cylinder is used for fdisk table */
961             dkg.dkg_ncyl = dkg.dkg_ncyl - 1;
962             media_size = (diskaddr_t)dkg.dkg_ncyl * dkg.dkg_nhead *
963                 dkg.dkg_nsect;
964         }
965     }

967 #ifdef DEBUG
968     DPRINTF1("Ncyl %d\n", dkg.dkg_ncyl);
969     DPRINTF1("nhead %d\n", dkg.dkg_nhead);
970     DPRINTF1("nsect %d\n", dkg.dkg_nsect);
971 #endif /* DEBUG */

973     if (media_size == 0) {
974         media_size = (uint32_t)med_info.sm_capacity;
975     }

977     (void) memset(&part, 0, sizeof (part));
978     for (i = 0, j = 0; i < NDKMAP; i++) {

```

```

979         if (vt->v_part[i].p_tag == V_BACKUP) {
980             if (vt->v_part[i].p_start != 0) {
981                 (void) fprintf(stderr,
982                     gettext(
983                         "Backup slice should start at sector 0\n"));
984                 return (-1);
985             }
986             backup_size = vt->v_part[i].p_size;
987             num_backup++;
988             continue;
989         }
990         if (vt->v_part[i].p_size) {

992             if (sparc_style) {
993                 if (vt->v_part[i].p_start % cyl_size) {
994                     (void) fprintf(stderr,
995                         gettext(
996                             "Slice %d does not start on cylinder boundary\n"), i);
997                     (void) fprintf(stderr,
998                         gettext(
999                             "Cylinder size %d 512 byte sectors\n"), cyl_size);
1000                     return (-1);
1001                 }
1002             }
1003             part[j].start = vt->v_part[i].p_start;
1004             part[j].end = vt->v_part[i].p_start +
1005                 vt->v_part[i].p_size - 1;
1006             part[j].num = i;
1007             j++;
1008         }
1009     }
1010     if (num_backup > 1) {
1011         (void) fprintf(stderr,
1012             gettext("Maximum one backup slice is allowed\n"));
1013         (void) smedia_release_handle(handle);
1014         (void) close(fd);
1015         exit(1);
1016     }
1017     num_slices = j;

1019     for (i = 0; i < num_slices; i++) {
1020         min_val = part[i].start;
1021         min_slice = i;
1022         for (j = i+1; j < num_slices; j++) {
1023             if (part[j].start < min_val) {
1024                 min_val = part[j].start;
1025                 min_slice = j;
1026             }
1027         }
1028         if (min_slice != i) {
1029             SWAP(part[i].start, part[min_slice].start)
1030             SWAP(part[i].end, part[min_slice].end)
1031             SWAP(part[i].num, part[min_slice].num)
1032         }
1033     }

1035 #ifdef DEBUG
1036     for (i = 0; i < num_slices; i++) {
1037         DPRINTF4("\n %d (%d) : %llu, %llu", i, part[i].num,
1038             part[i].start, part[i].end);
1039     }
1040 #endif /* DEBUG */

1042     if (backup_size > media_size) {
1043         if (sparc_style) {
1044             (void) fprintf(stderr,

```

```

1045         gettext(
1046             "Backup slice extends beyond size of media\n");
1047         (void) fprintf(stderr,
1048             gettext("media size : %llu sectors \n"),
1049             media_size);
1050     } else {
1051
1052         (void) fprintf(stderr,
1053             gettext("Backup slice extends beyond size of FDISK \
1054 Solaris partition\n"));
1055         (void) fprintf(stderr,
1056             gettext(
1057                 "FDISK Solaris partition size : %llu sectors \n"),
1058             media_size);
1059     }
1060     return (-1);
1061 }
1062
1063 /*
1064  * If we have only backup slice return success here.
1065  */
1066 if (num_slices == 0)
1067     return (0);
1068
1069 if (backup_size) {
1070     if (part[num_slices - 1].end > backup_size) {
1071         (void) fprintf(stderr,
1072             gettext("Slice %d extends beyond backup slice.\n"),
1073             part[num_slices - 1].num);
1074         return (-1);
1075     }
1076 } else {
1077     if (part[num_slices - 1].end > media_size) {
1078         if (sparc_style) {
1079             (void) fprintf(stderr,
1080                 gettext(
1081                     "Slice %d extends beyond media size\n"),
1082                 part[num_slices - 1].num);
1083             (void) fprintf(stderr,
1084                 gettext("media size : %llu sectors \n"),
1085                 media_size);
1086         } else {
1087             (void) fprintf(stderr,
1088                 gettext("Slice %d extends beyond FDISK"
1089                     " Solaris partition size\n"),
1090                 part[num_slices - 1].num);
1091             (void) fprintf(stderr, gettext(
1092                 "FDISK Solaris partition size : %llu "
1093                 "sectors \n"), media_size);
1094         }
1095         return (-1);
1096     }
1097 }
1098
1099 for (i = 0; i < num_slices; i++) {
1100     if (i == 0)
1101         continue;
1102     if (part[i].start <= part[i-1].end) {
1103         (void) fprintf(stderr,
1104             gettext("Overlap between slices %d and %d\n"),
1105             part[i-1].num, part[i].num);
1106         (void) smedia_release_handle(handle);
1107         (void) close(fd);
1108         exit(1);
1109     }
1110 }

```

```

1111     }
1112 }
1113
1114     return (0);
1115 }
1116
1117 unchanged_portion_omitted

```

new/usr/src/cmd/rmvolmgr/rmm_common.c

1

```
*****
33521 Sun May 4 18:28:36 2014
new/usr/src/cmd/rmvolmgr/rmm_common.c
4833 Remove volrmmount
4845 rm(u)mount don't always print mount/unmount errors
4846 HAL partition names don't match real partition names
Reviewed by: Dan McDonald <danmcd@omniti.com>
Reviewed by: Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 *
21 */
22 * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 *
25 * Copyright 2014 Andrew Stormont.
26 #endif /* ! codereview */
27 */
28
29 #include <stdio.h>
30 #include <errno.h>
31 #include <string.h>
32 #include <strings.h>
33 #include <stdarg.h>
34 #include <fcntl.h>
35 #include <libintl.h>
36 #include <stdlib.h>
37 #include <unistd.h>
38 #include <ctype.h>
39 #include <sys/param.h>
40 #include <sys/types.h>
41 #include <sys/stat.h>
42 #include <sys/mnttab.h>
43
44 #include <dbus/dbus.h>
45 #include <dbus/dbus-glib.h>
46 #include <dbus/dbus-glib-lowlevel.h>
47 #include <libhal.h>
48 #include <libhal-storage.h>
49
50 #include "rmm_common.h"
51
52 #define RMM_PRINT_DEVICE_WIDTH 20
53
54 extern int rmm_debug;
```

new/usr/src/cmd/rmvolmgr/rmm_common.c

2

```
56 static const char *action_strings[] = {
57     "eject",
58     "mount",
59     "remount",
60     "unmount",
61     "clear_mounts",
62     "closetray"
63 };
64
65
66 LibHalContext *
67 rmm_hal_init(LibHalDeviceAdded devadd_cb, LibHalDeviceRemoved devrem_cb,
68             LibHalDevicePropertyModified propmod_cb, LibHalDeviceCondition cond_cb,
69             DBusError *error, rmm_error_t *rmm_error)
70 {
71     DBusConnection *dbus_conn;
72     LibHalContext *ctx;
73     char **devices;
74     int nr;
75
76     dbus_error_init(error);
77
78     /*
79      * setup D-Bus connection
80      */
81     if (!(dbus_conn = dbus_bus_get(DBUS_BUS_SYSTEM, error))) {
82         dprintf("cannot get system bus: %s\n", rmm_strerror(error, -1));
83         *rmm_error = RMM_EDBUS_CONNECT;
84         return (NULL);
85     }
86     rmm_dbus_error_free(error);
87
88     dbus_connection_setup_with_g_main(dbus_conn, NULL);
89     dbus_connection_set_exit_on_disconnect(dbus_conn, B_TRUE);
90
91     if ((ctx = libhal_ctx_new()) == NULL) {
92         dprintf("libhal_ctx_new failed");
93         *rmm_error = RMM_EHAL_CONNECT;
94         return (NULL);
95     }
96
97     libhal_ctx_set_dbus_connection(ctx, dbus_conn);
98
99     /*
100      * register callbacks
101      */
102     if (devadd_cb != NULL) {
103         libhal_ctx_set_device_added(ctx, devadd_cb);
104     }
105     if (devrem_cb != NULL) {
106         libhal_ctx_set_device_removed(ctx, devrem_cb);
107     }
108     if (propmod_cb != NULL) {
109         libhal_ctx_set_device_property_modified(ctx, propmod_cb);
110         if (!libhal_device_property_watch_all(ctx, error)) {
111             dprintf("property_watch_all failed %s",
112                   rmm_strerror(error, -1));
113             libhal_ctx_free(ctx);
114             *rmm_error = RMM_EHAL_CONNECT;
115             return (NULL);
116         }
117     }
118     if (cond_cb != NULL) {
119         libhal_ctx_set_device_condition(ctx, cond_cb);
120     }
121 }
```

```

122     if (!libhal_ctx_init(ctx, error)) {
123         dprintf("libhal_ctx_init failed: %s", rmm_strerror(error, -1));
124         libhal_ctx_free(ctx);
125         *rmm_error = RMM_EHAL_CONNECT;
126         return (NULL);
127     }
128     rmm_dbus_error_free(error);

130     /*
131     * The above functions do not guarantee that HAL is actually running.
132     * Check by invoking a method.
133     */
134     if (!(devices = libhal_get_all_devices(ctx, &nr, error))) {
135         dprintf("HAL is not running: %s", rmm_strerror(error, -1));
136         libhal_ctx_shutdown(ctx, NULL);
137         libhal_ctx_free(ctx);
138         *rmm_error = RMM_EHAL_CONNECT;
139         return (NULL);
140     } else {
141         rmm_dbus_error_free(error);
142         libhal_free_string_array(devices);
143     }

145     return (ctx);
146 }

149 void
150 rmm_hal_fini(LibHalContext *hal_ctx)
151 {
152     DBusConnection *dbus_conn = libhal_ctx_get_dbus_connection(hal_ctx);

154     (void) dbus_connection_unref(dbus_conn);
155     (void) libhal_ctx_free(hal_ctx);
156 }

159 /*
160 * find volume from any type of name, similar to the old media_findname()
161 * returns the LibHalDrive object and a list of LibHalVolume objects.
162 */
163 LibHalDrive *
164 rmm_hal_volume_find(LibHalContext *hal_ctx, const char *name, DBusError *error,
165                    GSList **volumes)
166 {
167     LibHalDrive *drive;
168     char *p;
169     char lastc;

171     *volumes = NULL;

173     /* temporarily remove trailing slash */
174     p = (char *)name + strlen(name) - 1;
175     if (*p == '/') {
176         lastc = *p;
177         *p = '\0';
178     } else {
179         p = NULL;
180     }

182     if (name[0] == '/') {
183         if (((drive = rmm_hal_volume_findby(hal_ctx,
184         "info.udi", name, volumes)) != NULL) ||
185         ((drive = rmm_hal_volume_findby(hal_ctx,
186         "block.device", name, volumes)) != NULL) ||
187         ((drive = rmm_hal_volume_findby(hal_ctx,

```

```

188         "block.solaris.raw_device", name, volumes)) != NULL) ||
189         ((drive = rmm_hal_volume_findby(hal_ctx,
190         "volume.mount_point", name, volumes)) != NULL)) {
191             goto out;
192         } else {
193             goto out;
194         }
195     }

197     /* try volume label */
198     if ((drive = rmm_hal_volume_findby(hal_ctx,
199     "volume.label", name, volumes)) != NULL) {
200         goto out;
201     }

203     drive = rmm_hal_volume_findby_nickname(hal_ctx, name, volumes);

205 out:
206     if (p != NULL) {
207         *p = lastc;
208     }
209     return (drive);
210 }

212 /*
213 * find default volume. Returns volume pointer and name in 'name'.
214 */
215 LibHalDrive *
216 rmm_hal_volume_find_default(LibHalContext *hal_ctx, DBusError *error,
217                             const char **name_out, GSList **volumes)
218 {
219     LibHalDrive *drive;
220     static const char *names[] = { "floppy", "cdrom", "rmdisk" };
221     int i;

223     *volumes = NULL;

225     for (i = 0; i < NELEM(names); i++) {
226         if ((drive = rmm_hal_volume_findby_nickname(hal_ctx,
227         names[i], volumes)) != NULL) {
228             /*
229             * Skip floppy if it has no media.
230             * XXX might want to actually check for media
231             * every time instead of relying on volcheck.
232             */
233             if ((strcmp(names[i], "floppy") != 0) ||
234             libhal_device_get_property_bool(hal_ctx,
235             libhal_drive_get_udi(drive),
236             "storage.removable.media_available", NULL)) {
237                 *name_out = names[i];
238                 break;
239             }
240         }
241         rmm_dbus_error_free(error);
242     }

244     return (drive);
245 }

247 /*
248 * find volume by property=value
249 * returns the LibHalDrive object and a list of LibHalVolume objects.
250 * XXX add support for multiple properties, reduce D-Bus traffic
251 */
252 LibHalDrive *
253 rmm_hal_volume_findby(LibHalContext *hal_ctx, const char *property,

```

```

254     const char *value, GSList **volumes)
255 {
256     DBusError      error;
257     LibHalDrive    *drive = NULL;
258     LibHalVolume   *v = NULL;
259     char           **udis;
260     int            num_udis;
261     int            i;
262     int            i_drive = -1;
263
264     *volumes = NULL;
265
266     dbus_error_init(&error);
267
268     /* get all devices with property=value */
269     if ((udis = libhal_manager_find_device_string_match(hal_ctx, property,
270     value, &num_udis, &error)) == NULL) {
271         rmm_dbus_error_free(&error);
272         return (NULL);
273     }
274
275     /* find volumes and drives among these devices */
276     for (i = 0; i < num_udis; i++) {
277         rmm_dbus_error_free(&error);
278         if (libhal_device_query_capability(hal_ctx, udis[i], "volume",
279     &error)) {
280             v = libhal_volume_from_udi(hal_ctx, udis[i]);
281             if (v != NULL) {
282                 *volumes = g_slist_prepend(*volumes, v);
283             }
284         } else if ((*volumes == NULL) &&
285     libhal_device_query_capability(hal_ctx, udis[i], "storage",
286     &error)) {
287             i_drive = i;
288         }
289     }
290
291     if (*volumes != NULL) {
292         /* used prepend, preserve original order */
293         *volumes = g_slist_reverse(*volumes);
294
295         v = (LibHalVolume *)(*volumes)->data;
296         drive = libhal_drive_from_udi(hal_ctx,
297     libhal_volume_get_storage_device_udi(v));
298         if (drive == NULL) {
299             rmm_volumes_free (*volumes);
300             *volumes = NULL;
301         }
302     } else if (i_drive >= 0) {
303         drive = libhal_drive_from_udi(hal_ctx, udis[i_drive]);
304     }
305
306     libhal_free_string_array(udis);
307     rmm_dbus_error_free(&error);
308
309     return (drive);
310 }
311
312 static void
313 rmm_print_nicknames_one(LibHalDrive *d, LibHalVolume *v,
314     const char *device, char **drive_nicknames)
315 {
316     const char      *volume_label = NULL;
317     const char      *mount_point = NULL;
318     boolean_t       comma;
319     int             i;

```

```

321     (void) printf("%-*s ", RMM_PRINT_DEVICE_WIDTH, device);
322     comma = B_FALSE;
323
324     if (drive_nicknames != NULL) {
325         for (i = 0; drive_nicknames[i] != NULL; i++) {
326             (void) printf("%s%s", comma ? ", " : "",
327     drive_nicknames[i]);
328             comma = B_TRUE;
329         }
330     }
331
332     if ((v != NULL) &&
333     ((volume_label = libhal_volume_get_label(v)) != NULL) &&
334     (strlen(volume_label) > 0)) {
335         (void) printf("%s%s", comma ? ", " : "", volume_label);
336         comma = B_TRUE;
337     }
338
339     if ((v != NULL) &&
340     ((mount_point = libhal_volume_get_mount_point(v)) != NULL) &&
341     (strlen(mount_point) > 0)) {
342         (void) printf("%s%s", comma ? ", " : "", mount_point);
343         comma = B_TRUE;
344     }
345
346     (void) printf("\n");
347 }
348
349 /*
350  * print nicknames for each available volume
351  *
352  * print_mask:
353  *   RMM_PRINT_MOUNTABLE      print only mountable volumes
354  *   RMM_PRINT_EJECTABLE     print volume-less ejectable drives
355  */
356 void
357 rmm_print_volume_nicknames(LibHalContext *hal_ctx, DBusError *error,
358     int print_mask)
359 {
360     char           **udis;
361     int            num_udis;
362     GSList         *volumes = NULL;
363     LibHalDrive    *d, *d_tmp;
364     LibHalVolume   *v;
365     const char     *device;
366     char           **nicknames;
367     int            i;
368     GSList         *j;
369     int            nprinted;
370
371     dbus_error_init(error);
372
373     if ((udis = libhal_find_device_by_capability(hal_ctx, "storage",
374     &num_udis, error)) == NULL) {
375         rmm_dbus_error_free(error);
376         return;
377     }
378
379     for (i = 0; i < num_udis; i++) {
380         if ((d = libhal_drive_from_udi(hal_ctx, udis[i])) == NULL) {
381             continue;
382         }
383
384         /* find volumes belonging to this drive */
385         if ((d_tmp = rmm_hal_volume_findby(hal_ctx,

```

```

386         "block.storage_device", udis[i], &volumes)) != NULL) {
387             libhal_drive_free(d_tmp);
388         }
390     nicknames = libhal_device_get_property_strlist(hal_ctx,
391         udis[i], "storage.solaris.nicknames", NULL);
393     nprinted = 0;
394     for (j = volumes; j != NULL; j = g_slist_next(j)) {
395         v = (LibHalVolume *) (j->data);
397         if ((device = libhal_volume_get_device_file(v)) ==
398             NULL) {
399             continue;
400         }
401         if ((print_mask & RMM_PRINT_MOUNTABLE) &&
402             (libhal_volume_get_fsusage(v) !=
403              LIBHAL_VOLUME_USAGE_MOUNTABLE_FILESYSTEM)) {
404             continue;
405         }
407         rmm_print_nicknames_one(d, v, device, nicknames);
408         nprinted++;
409     }
411     if ((nprinted == 0) &&
412         (print_mask & RMM_PRINT_EJECTABLE) &&
413         libhal_drive_requires_eject(d) &&
414         ((device = libhal_drive_get_device_file(d)) != NULL)) {
415         rmm_print_nicknames_one(d, NULL, device, nicknames);
416     }
418     libhal_free_string_array(nicknames);
419     libhal_drive_free(d);
420     rmm_volumes_free(volumes);
421     volumes = NULL;
422 }
424 libhal_free_string_array(udis);
425 }
427 /*
428  * find volume by nickname
429  * returns the LibHalDrive object and a list of LibHalVolume objects.
430  */
431 LibHalDrive *
432 rmm_hal_volume_findby_nickname(LibHalContext *hal_ctx, const char *name,
433     GSList **volumes)
434 {
435     DBusError    error;
436     LibHalDrive *drive = NULL;
437     LibHalDrive *drive_tmp;
438     char         **udis;
439     int          num_udis;
440     char         **nicknames;
441     int          i, j;
443     *volumes = NULL;
445     dbus_error_init(&error);
447     if ((udis = libhal_find_device_by_capability(hal_ctx, "storage",
448         &num_udis, &error)) == NULL) {
449         rmm_dbus_error_free(&error);
450         return (NULL);
451     }

```

```

453     /* find a drive by nickname */
454     for (i = 0; (i < num_udis) && (drive == NULL); i++) {
455         if ((nicknames = libhal_device_get_property_strlist(hal_ctx,
456             udis[i], "storage.solaris.nicknames", &error)) == NULL) {
457             rmm_dbus_error_free(&error);
458             continue;
459         }
460         for (j = 0; (nicknames[j] != NULL) && (drive == NULL); j++) {
461             if (strcmp(nicknames[j], name) == 0) {
462                 drive = libhal_drive_from_udi(hal_ctx, udis[i]);
463             }
464         }
465         libhal_free_string_array(nicknames);
466     }
467     libhal_free_string_array(udis);
469     if (drive != NULL) {
470         /* found the drive, now find its volumes */
471         if ((drive_tmp = rmm_hal_volume_findby(hal_ctx,
472             "block.storage_device", libhal_drive_get_udi(drive),
473             volumes)) != NULL) {
474             libhal_drive_free(drive_tmp);
475         }
476     }
478     rmm_dbus_error_free(&error);
480     return (drive);
481 }
483 void
484 rmm_volumes_free(GSList *volumes)
485 {
486     GSList *i;
488     for (i = volumes; i != NULL; i = g_slist_next(i)) {
489         libhal_volume_free((LibHalVolume *) (i->data));
490     }
491     g_slist_free(volumes);
492 }
494 /*
495  * Call HAL's Mount() method on the given device
496  */
497 boolean_t
498 rmm_hal_mount(LibHalContext *hal_ctx, const char *udi,
499     char **opts, int num_opts, char *mountpoint, DBusError *error)
500 {
501     DBusConnection *dbus_conn = libhal_ctx_get_dbus_connection(hal_ctx);
502     DBusMessage     *dmsg, *reply;
503     char             *fstype;
505     dprintf("mounting %s...\n", udi);
507     if (!(dmsg = dbus_message_new_method_call("org.freedesktop.Hal", udi,
508         "org.freedesktop.Hal.Device.Volume", "Mount"))) {
509         dprintf(
510             "mount failed for %s: cannot create dbus message\n", udi);
511         return (B_FALSE);
512     }
514     fstype = "";
515     if (mountpoint == NULL) {
516         mountpoint = "";
517     }

```

```

519     if (!dbus_message_append_args(dmesg, DBUS_TYPE_STRING, &mountpoint,
520     DBUS_TYPE_STRING, &fstype,
521     DBUS_TYPE_ARRAY, DBUS_TYPE_STRING, &opts, num_opts,
522     DBUS_TYPE_INVALID)) {
523         dprintf("mount failed for %s: cannot append args\n", udi);
524         dbus_message_unref(dmesg);
525         return (B_FALSE);
526     }
528     dbus_error_init(error);
529     if (!(reply = dbus_connection_send_with_reply_and_block(dbus_conn,
530     dmesg, RMM_MOUNT_TIMEOUT, error))) {
531         dprintf("mount failed for %s: %s\n", udi, error->message);
532         dbus_message_unref(dmesg);
533         return (B_FALSE);
534     }
536     dprintf("mounted %s\n", udi);
538     dbus_message_unref(dmesg);
539     dbus_message_unref(reply);
541     rmm_dbus_error_free(error);
543     return (B_TRUE);
544 }
547 /*
548  * Call HAL's Unmount() method on the given device
549  */
550 boolean_t
551 rmm_hal_unmount(LibHalContext *hal_ctx, const char *udi, DBusError *error)
552 {
553     DBusConnection *dbus_conn = libhal_ctx_get_dbus_connection(hal_ctx);
554     DBusMessage *dmesg, *reply;
555     char **opts = NULL;
557     dprintf("unmounting %s...\n", udi);
559     if (!(dmesg = dbus_message_new_method_call("org.freedesktop.Hal", udi,
560     "org.freedesktop.Hal.Device.Volume", "Unmount"))) {
561         dprintf(
562             "unmount failed %s: cannot create dbus message\n", udi);
563         return (B_FALSE);
564     }
566     if (!dbus_message_append_args(dmesg, DBUS_TYPE_ARRAY, DBUS_TYPE_STRING,
567     &opts, 0, DBUS_TYPE_INVALID)) {
568         dprintf("unmount failed %s: cannot append args\n", udi);
569         dbus_message_unref(dmesg);
570         return (B_FALSE);
571     }
573     dbus_error_init(error);
574     if (!(reply = dbus_connection_send_with_reply_and_block(dbus_conn,
575     dmesg, RMM_UNMOUNT_TIMEOUT, error))) {
576         dprintf("unmount failed for %s: %s\n", udi, error->message);
577         dbus_message_unref(dmesg);
578         return (B_FALSE);
579     }
581     dprintf("unmounted %s\n", udi);
583     dbus_message_unref(dmesg);

```

```

584     dbus_message_unref(reply);
586     rmm_dbus_error_free(error);
588     return (B_TRUE);
589 }
592 /*
593  * Call HAL's Eject() method on the given device
594  */
595 boolean_t
596 rmm_hal_eject(LibHalContext *hal_ctx, const char *udi, DBusError *error)
597 {
598     DBusConnection *dbus_conn = libhal_ctx_get_dbus_connection(hal_ctx);
599     DBusMessage *dmesg, *reply;
600     char **options = NULL;
601     uint_t num_options = 0;
603     dprintf("ejecting %s...\n", udi);
605     if (!(dmesg = dbus_message_new_method_call("org.freedesktop.Hal", udi,
606     "org.freedesktop.Hal.Device.Storage", "Eject"))) {
607         dprintf("eject %s: cannot create dbus message\n", udi);
608         return (B_FALSE);
609     }
611     if (!dbus_message_append_args(dmesg,
612     DBUS_TYPE_ARRAY, DBUS_TYPE_STRING, &options, num_options,
613     DBUS_TYPE_INVALID)) {
614         dprintf("eject %s: cannot append args to dbus message ", udi);
615         dbus_message_unref(dmesg);
616         return (B_FALSE);
617     }
619     dbus_error_init(error);
620     if (!(reply = dbus_connection_send_with_reply_and_block(dbus_conn,
621     dmesg, RMM_EJECT_TIMEOUT, error))) {
622         dprintf("eject %s: %s\n", udi, error->message);
623         dbus_message_unref(dmesg);
624         return (B_FALSE);
625     }
627     dprintf("ejected %s\n", udi);
629     dbus_message_unref(dmesg);
630     dbus_message_unref(reply);
632     rmm_dbus_error_free(error);
634     return (B_TRUE);
635 }
637 /*
638  * Call HAL's CloseTray() method on the given device
639  */
640 boolean_t
641 rmm_hal_closetray(LibHalContext *hal_ctx, const char *udi, DBusError *error)
642 {
643     DBusConnection *dbus_conn = libhal_ctx_get_dbus_connection(hal_ctx);
644     DBusMessage *dmesg, *reply;
645     char **options = NULL;
646     uint_t num_options = 0;
648     dprintf("closing tray %s...\n", udi);

```

```

650     if (!(dmesg = dbus_message_new_method_call("org.freedesktop.Hal", udi,
651         "org.freedesktop.Hal.Device.Storage", "CloseTray"))) {
652         dprintf(
653             "closetray failed for %s: cannot create dbus message\n",
654             udi);
655         return (B_FALSE);
656     }

658     if (!dbus_message_append_args(dmesg,
659         DBUS_TYPE_ARRAY, DBUS_TYPE_STRING, &options, num_options,
660         DBUS_TYPE_INVALID)) {
661         dprintf("closetray %s: cannot append args to dbus message ",
662             udi);
663         dbus_message_unref(dmesg);
664         return (B_FALSE);
665     }

667     dbus_error_init(&error);
668     if (!(reply = dbus_connection_send_with_reply_and_block(dbus_conn,
669         dmesg, RMM_CLOSETRAY_TIMEOUT, &error))) {
670         dprintf("closetray failed for %s: %s\n", udi, error->message);
671         dbus_message_unref(dmesg);
672         return (B_FALSE);
673     }

675     dprintf("closetray ok %s\n", udi);

677     dbus_message_unref(dmesg);
678     dbus_message_unref(reply);

680     rmm_dbus_error_free(&error);

682     return (B_TRUE);
683 }

685 /*
686  * Call HAL's Rescan() method on the given device
687  */
688 boolean_t
689 rmm_hal_rescan(LibHalContext *hal_ctx, const char *udi, DBusError *error)
690 {
691     DBusConnection *dbus_conn = libhal_ctx_get_dbus_connection(hal_ctx);
692     DBusMessage *dmesg, *reply;

694     dprintf("rescanning %s...\n", udi);

696     if (!(dmesg = dbus_message_new_method_call("org.freedesktop.Hal", udi,
697         "org.freedesktop.Hal.Device", "Rescan"))) {
698         dprintf("rescan failed for %s: cannot create dbus message\n",
699             udi);
700         return (B_FALSE);
701     }

703     dbus_error_init(&error);
704     if (!(reply = dbus_connection_send_with_reply_and_block(dbus_conn,
705         dmesg, -1, &error))) {
706         dprintf("rescan failed for %s: %s\n", udi, error->message);
707         dbus_message_unref(dmesg);
708         return (B_FALSE);
709     }

711     dprintf("rescan ok %s\n", udi);

713     dbus_message_unref(dmesg);
714     dbus_message_unref(reply);

```

```

716     rmm_dbus_error_free(&error);

718     return (B_TRUE);
719 }

721 boolean_t
722 rmm_hal_claim_branch(LibHalContext *hal_ctx, const char *udi)
723 {
724     DBusError error;
725     DBusConnection *dbus_conn = libhal_ctx_get_dbus_connection(hal_ctx);
726     DBusMessage *dmesg, *reply;
727     const char *claimed_by = "rmvolmgr";

729     dprintf("claiming branch %s...\n", udi);

731     if (!(dmesg = dbus_message_new_method_call("org.freedesktop.Hal",
732         "/org/freedesktop/Hal/Manager", "org.freedesktop.Hal.Manager",
733         "ClaimBranch"))) {
734         dprintf("cannot create dbus message\n");
735         return (B_FALSE);
736     }

738     if (!dbus_message_append_args(dmesg, DBUS_TYPE_STRING, &udi,
739         DBUS_TYPE_STRING, &claimed_by, DBUS_TYPE_INVALID)) {
740         dprintf("cannot append args to dbus message\n");
741         dbus_message_unref(dmesg);
742         return (B_FALSE);
743     }

745     dbus_error_init(&error);
746     if (!(reply = dbus_connection_send_with_reply_and_block(dbus_conn,
747         dmesg, -1, &error))) {
748         dprintf("cannot send dbus message\n");
749         dbus_message_unref(dmesg);
750         rmm_dbus_error_free(&error);
751         return (B_FALSE);
752     }

754     dprintf("claim branch ok %s\n", udi);

756     dbus_message_unref(dmesg);
757     dbus_message_unref(reply);

759     return (B_TRUE);
760 }

762 boolean_t
763 rmm_hal_unclaim_branch(LibHalContext *hal_ctx, const char *udi)
764 {
765     DBusError error;
766     DBusConnection *dbus_conn = libhal_ctx_get_dbus_connection(hal_ctx);
767     DBusMessage *dmesg, *reply;
768     const char *claimed_by = "rmvolmgr";

770     dprintf("unclaiming branch %s...\n", udi);

772     if (!(dmesg = dbus_message_new_method_call("org.freedesktop.Hal",
773         "/org/freedesktop/Hal/Manager", "org.freedesktop.Hal.Manager",
774         "UnclaimBranch"))) {
775         dprintf("cannot create dbus message\n");
776         return (B_FALSE);
777     }

779     if (!dbus_message_append_args(dmesg, DBUS_TYPE_STRING, &udi,
780         DBUS_TYPE_STRING, &claimed_by, DBUS_TYPE_INVALID)) {
781         dprintf("cannot append args to dbus message\n");

```



```

782     dbus_message_unref(dmsg);
783     return (B_FALSE);
784 }

786     dbus_error_init(&error);
787     if (!(reply = dbus_connection_send_with_reply_and_block(dbus_conn,
788     dmsg, -1, &error))) {
789         dprintf("cannot send dbus message\n");
790         dbus_message_unref(dmsg);
791         rmm_dbus_error_free(&error);
792         return (B_FALSE);
793     }

795     dprintf("unclaim branch ok %s\n", udi);

797     dbus_message_unref(dmsg);
798     dbus_message_unref(reply);

800     return (B_TRUE);
801 }

803 static boolean_t
804 rmm_action_one(LibHalContext *hal_ctx, const char *name, action_t action,
805     const char *dev, const char *udi, LibHalVolume *v,
806     char **opts, int num_opts, char *mountpoint)
807 {
808     char        dev_str[MAXPATHLEN];
809     char        *mountp;
810     DBusError    error;
811     boolean_t    ret = B_FALSE;

813     dprintf("rmm_action_one %s %s\n", name, action_strings[action]);

815 #endif /* ! codereview */
816     if (strcmp(name, dev) == 0) {
817         (void) snprintf(dev_str, sizeof (dev_str), name);
818     } else {
819         (void) snprintf(dev_str, sizeof (dev_str), "%s %s", name, dev);
820     }

822     dbus_error_init(&error);

824     switch (action) {
825     case EJECT:
826         ret = rmm_hal_eject(hal_ctx, udi, &error);
827         break;
828     case INSERT:
829     case REMOUNT:
830         if (libhal_volume_is_mounted(v)) {
831             goto done;
832         }
830         ret = rmm_hal_mount(hal_ctx, udi,
831             opts, num_opts, mountpoint, &error);
832         break;
833     case UNMOUNT:
834         if (!libhal_volume_is_mounted(v)) {
835             goto done;
836         }
834         ret = rmm_hal_unmount(hal_ctx, udi, &error);
835         break;
836     case CLOSETRAY:
837         ret = rmm_hal_closetray(hal_ctx, udi, &error);
838         break;
839     }

841     if (!ret) {

```

```

842         (void) fprintf(stderr, gettext("%s of %s failed: %s\n"),
843             action_strings[action], dev_str, rmm_strerror(&error, -1));
844         goto done;
845     }

847     switch (action) {
848     case EJECT:
849         (void) printf(gettext("%s ejected\n"), dev_str);
850         break;
851     case INSERT:
852     case REMOUNT:
853         mountp = rmm_get_mnttab_mount_point(dev);
854         if (mountp != NULL) {
855             (void) printf(gettext("%s mounted at %s\n"),
856                 dev_str, mountp);
857             free(mountp);
858         }
859         break;
860     case UNMOUNT:
861         (void) printf(gettext("%s unmounted\n"), dev_str);
862         break;
863     case CLOSETRAY:
864         (void) printf(gettext("%s tray closed\n"), dev_str);
865         break;
866     }

868 done:
869     rmm_dbus_error_free(&error);
870     return (ret);
871 }

873 /*
874  * top level action routine
875  *
876  * If non-null 'aa' is passed, it will be used, otherwise a local copy
877  * will be created.
878  */
879 boolean_t
880 rmm_action(LibHalContext *hal_ctx, const char *name, action_t action,
881     struct action_arg *aap, char **opts, int num_opts, char *mountpoint)
882 {
883     DBusError    error;
884     GSList       *volumes, *i;
885     LibHalDrive  *d;
886     LibHalVolume *v;
887     const char   *udi, *d_udi;
888     const char   *dev, *d_dev;
889     struct action_arg aa_local;
890     boolean_t    ret = B_FALSE;

892     dprintf("rmm_action %s %s\n", name, action_strings[action]);

894     if (aap == NULL) {
895         bzero(&aa_local, sizeof (aa_local));
896         aap = &aa_local;
897     }

899     dbus_error_init(&error);

901     /* find the drive and its volumes */
902     d = rmm_hal_volume_find(hal_ctx, name, &error, &volumes);
903     rmm_dbus_error_free(&error);
904     if (d == NULL) {
905         (void) fprintf(stderr, gettext("cannot find '%s'\n"), name);
906         return (B_FALSE);
907     }

```

```

908     d_udi = libhal_drive_get_udi(d);
909     d_dev = libhal_drive_get_device_file(d);
910     if ((d_udi == NULL) || (d_dev == NULL)) {
911         goto out;
912     }
913
914     /*
915     * For those drives that do not require media eject,
916     * EJECT turns into UNMOUNT.
917     */
918     if ((action == EJECT) && !libhal_drive_requires_eject(d)) {
919         action = UNMOUNT;
920     }
921
922     /*
923     * We can't mount or unmount a drive that has no volumes.
924     * Either the media isn't inserted or it's not formatted
925     */
926     if (volumes == NULL && (action != EJECT && action != CLOSETRAY)) {
927         (void) fprintf(stderr, libhal_drive_requires_eject(d) ?
928             gettext("no volumes in '%s' to %s\n") :
929             gettext("no volumes on '%s' to %s\n"),
930             name, action_strings[action]);
931         goto out;
932     }
933
934 #endif /* ! codereview */
935     /* per drive action */
936     if ((action == EJECT) || (action == CLOSETRAY)) {
937         ret = rmm_action_one(hal_ctx, name, action, d_dev, d_udi, NULL,
938             opts, num_opts, NULL);
939
940         if (!ret || (action == CLOSETRAY)) {
941             goto out;
942         }
943     }
944
945     /* per volume action */
946     for (i = volumes; i != NULL; i = g_slist_next(i)) {
947         v = (LibHalVolume *)i->data;
948         udi = libhal_volume_get_udi(v);
949         dev = libhal_volume_get_device_file(v);
950
951         if ((udi == NULL) || (dev == NULL)) {
952             continue;
953         }
954         if (aap == &aa_local) {
955             if (!rmm_volume_aa_from_prop(hal_ctx, udi, v, aap)) {
956                 dprintf("rmm_volume_aa_from_prop failed %s\n",
957                     udi);
958                 continue;
959             }
960         }
961         aap->aa_action = action;
962
963         /* ejected above, just need postprocess */
964         if (action != EJECT) {
965             ret = rmm_action_one(hal_ctx, name, action, dev, udi, v,
966                 opts, num_opts, mountpoint);
967         }
968         if (ret) {
969             (void) void_postprocess(hal_ctx, udi, aap);
970         }
971
972         if (aap == &aa_local) {
973             rmm_volume_aa_free(aap);

```

```

974     }
975 }
976
977 out:
978     if (volumes != NULL)
979 #endif /* ! codereview */
980         rmm_volumes_free(volumes);
981     if (d != NULL)
982 #endif /* ! codereview */
983         libhal_drive_free(d);
984
985     return (ret);
986 }
987
988 /*
989 * rescan by name
990 * if name is NULL, rescan all drives
991 */
992 #endif /* ! codereview */
993 boolean_t
994 rmm_rescan(LibHalContext *hal_ctx, const char *name, boolean_t query)
995 {
996     DBusError      error;
997     GSList         *volumes;
998     LibHalDrive    *drive = NULL;
999     const char     *drive_udi;
1000     char           **udis;
1001     int            num_udis;
1002     char           *nickname;
1003     char           **nicks = NULL;
1004     boolean_t      do_free_udis = FALSE;
1005     int            i;
1006     boolean_t      ret = B_FALSE;
1007
1008     dprintf("rmm_rescan %s\n", name != NULL ? name : "all");
1009
1010     dbus_error_init(&error);
1011
1012     if (name != NULL) {
1013         if ((drive = rmm_hal_volume_find(hal_ctx, name, &error,
1014             &volumes)) == NULL) {
1015             rmm_dbus_error_free(&error);
1016             (void) fprintf(stderr,
1017                 gettext("cannot find '%s'\n"), name);
1018             return (B_FALSE);
1019         }
1020         rmm_dbus_error_free(&error);
1021         g_slist_free(volumes);
1022
1023         drive_udi = libhal_drive_get_udi(drive);
1024         udis = (char **)&drive_udi;
1025         num_udis = 1;
1026     } else {
1027         if ((udis = libhal_find_device_by_capability(hal_ctx,
1028             "storage", &num_udis, &error)) == NULL) {
1029             rmm_dbus_error_free(&error);
1030             return (B_TRUE);
1031         }
1032         rmm_dbus_error_free(&error);
1033         do_free_udis = TRUE;
1034     }
1035
1036     for (i = 0; i < num_udis; i++) {
1037         if (name == NULL) {
1038             nicks = libhal_device_get_property_strlist(hal_ctx,
1039                 udis[i], "storage.solaris.nicknames", NULL);

```

```

1040         if (nicks != NULL) {
1041             nickname = nicks[0];
1042         } else {
1043             nickname = "";
1044         }
1045     }
1046     if (!(ret = rmm_hal_rescan(hal_ctx, udis[i], &error))) {
1047         (void) fprintf(stderr,
1048             gettext("rescan of %s failed: %s\n"),
1049             name ? name : nickname,
1050             rmm_strerror(&error, -1));
1051         libhal_free_string_array(nicks);
1052         continue;
1053     }
1054     if (query) {
1055         ret = libhal_device_get_property_bool(hal_ctx, udis[i],
1056             "storage.removable.media_available", NULL);
1057         if (ret) {
1058             printf(gettext("%s is available\n"),
1059                 name ? name : nickname);
1060         } else {
1061             printf(gettext("%s is not available\n"),
1062                 name ? name : nickname);
1063         }
1064     }
1065     libhal_free_string_array(nicks);
1066 }
1067
1068 if (drive != NULL) {
1069     libhal_drive_free(drive);
1070 }
1071 if (do_free_udis) {
1072     libhal_free_string_array(udis);
1073 }
1074
1075 return (ret);
1076 }
1077
1078 /*
1079  * set action_arg from volume properties
1080  */
1081 boolean_t
1082 rmm_volume_aa_from_prop(LibHalContext *hal_ctx, const char *udi_arg,
1083     LibHalVolume *volume_arg, struct action_arg *aap)
1084 {
1085     LibHalVolume *volume = volume_arg;
1086     const char *udi = udi_arg;
1087     const char *drive_udi;
1088     char *volume_label;
1089     char *mountpoint;
1090     int len;
1091     int ret = B_FALSE;
1092
1093     /* at least udi or volume must be supplied */
1094     if ((udi == NULL) && (volume == NULL)) {
1095         return (B_FALSE);
1096     }
1097     if (volume == NULL) {
1098         if ((volume = libhal_volume_from_udi(hal_ctx, udi)) == NULL) {
1099             dprintf("cannot get volume %s\n", udi);
1100             goto out;
1101         }
1102     }
1103     if (udi == NULL) {
1104         if ((udi = libhal_volume_get_udi(volume)) == NULL) {

```

```

1106             dprintf("cannot get udi\n");
1107             goto out;
1108         }
1109     }
1110     drive_udi = libhal_volume_get_storage_device_udi(volume);
1111
1112     if (!(aap->aa_symdev = libhal_device_get_property_string(hal_ctx,
1113         drive_udi, "storage.solaris.legacy.symdev", NULL))) {
1114         dprintf("property %s not found %s\n",
1115             "storage.solaris.legacy.symdev", drive_udi);
1116         goto out;
1117     }
1118     if (!(aap->aa_media = libhal_device_get_property_string(hal_ctx,
1119         drive_udi, "storage.solaris.legacy.media_type", NULL))) {
1120         dprintf("property %s not found %s\n",
1121             "storage.solaris.legacy.media_type", drive_udi);
1122         goto out;
1123     }
1124
1125     /* name is derived from volume label */
1126     aap->aa_name = NULL;
1127     if ((volume_label = (char *)libhal_device_get_property_string(hal_ctx,
1128         udi, "volume.label", NULL)) != NULL) {
1129         if ((len = strlen(volume_label)) > 0) {
1130             aap->aa_name = rmm_vold_convert_volume_label(
1131                 volume_label, len);
1132             if (strlen(aap->aa_name) == 0) {
1133                 free(aap->aa_name);
1134                 aap->aa_name = NULL;
1135             }
1136         }
1137         libhal_free_string(volume_label);
1138     }
1139     /* if no label, then unnamed_<mediatype> */
1140     if (aap->aa_name == NULL) {
1141         aap->aa_name = (char *)calloc(1, sizeof("unnamed_floppyNNNN"));
1142         if (aap->aa_name == NULL) {
1143             goto out;
1144         }
1145         (void) snprintf(aap->aa_name, sizeof("unnamed_floppyNNNN"),
1146             "unnamed_%s", aap->aa_media);
1147     }
1148
1149     if (!(aap->aa_path = libhal_device_get_property_string(hal_ctx, udi,
1150         "block.device", NULL))) {
1151         dprintf("property %s not found %s\n", "block.device", udi);
1152         goto out;
1153     }
1154     if (!(aap->aa_rawpath = libhal_device_get_property_string(hal_ctx, udi,
1155         "block.solaris.raw_device", NULL))) {
1156         dprintf("property %s not found %s\n",
1157             "block.solaris.raw_device", udi);
1158         goto out;
1159     }
1160     if (!(aap->aa_type = libhal_device_get_property_string(hal_ctx, udi,
1161         "volume.fstype", NULL))) {
1162         dprintf("property %s not found %s\n", "volume.fstype", udi);
1163         goto out;
1164     }
1165     if (!libhal_device_get_property_bool(hal_ctx, udi,
1166         "volume.is_partition", NULL)) {
1167         aap->aa_partname = NULL;
1168     } else if (!(aap->aa_partname = libhal_device_get_property_string(
1169         hal_ctx, udi, "block.solaris.slice", NULL))) {
1170         dprintf("property %s not found %s\n",
1171             "block.solaris.slice", udi);

```

```

1172     goto out;
1173 }
1174 if (!(mountpoint = libhal_device_get_property_string(hal_ctx, udi,
1175     "volume.mount_point", NULL)) {
1176     dprintf("property %s not found %s\n",
1177     "volume.mount_point", udi);
1178     goto out;
1179 }
1180 /*
1181  * aa_mountpoint can be reallocated in rmm_volume_aa_update_mountpoint()
1182  * won't have to choose between free() or libhal_free_string() later on
1183  */
1184 aap->aa_mountpoint = strdup(mountpoint);
1185 libhal_free_string(mountpoint);
1186 if (aap->aa_mountpoint == NULL) {
1187     dprintf("mountpoint is NULL %s\n", udi);
1188     goto out;
1189 }
1191 ret = B_TRUE;
1193 out:
1194 if ((volume != NULL) && (volume != volume_arg)) {
1195     libhal_volume_free(volume);
1196 }
1197 if (!ret) {
1198     rmm_volume_aa_free(aap);
1199 }
1200 return (ret);
1201 }
1203 /* ARGSUSED */
1204 void
1205 rmm_volume_aa_update_mountpoint(LibHalContext *hal_ctx, const char *udi,
1206     struct action_arg *aap)
1207 {
1208     if (aap->aa_mountpoint != NULL) {
1209         free(aap->aa_mountpoint);
1210     }
1211     aap->aa_mountpoint = rmm_get_mnttab_mount_point(aap->aa_path);
1212 }
1214 void
1215 rmm_volume_aa_free(struct action_arg *aap)
1216 {
1217     if (aap->aa_symdev != NULL) {
1218         libhal_free_string(aap->aa_symdev);
1219         aap->aa_symdev = NULL;
1220     }
1221     if (aap->aa_name != NULL) {
1222         free(aap->aa_name);
1223         aap->aa_name = NULL;
1224     }
1225     if (aap->aa_path != NULL) {
1226         libhal_free_string(aap->aa_path);
1227         aap->aa_path = NULL;
1228     }
1229     if (aap->aa_rawpath != NULL) {
1230         libhal_free_string(aap->aa_rawpath);
1231         aap->aa_rawpath = NULL;
1232     }
1233     if (aap->aa_type != NULL) {
1234         libhal_free_string(aap->aa_type);
1235         aap->aa_type = NULL;
1236     }
1237     if (aap->aa_media != NULL) {

```

```

1238     libhal_free_string(aap->aa_media);
1239     aap->aa_media = NULL;
1240 }
1241 if (aap->aa_partname != NULL) {
1242     libhal_free_string(aap->aa_partname);
1243     aap->aa_partname = NULL;
1244 }
1245 if (aap->aa_mountpoint != NULL) {
1246     free(aap->aa_mountpoint);
1247     aap->aa_mountpoint = NULL;
1248 }
1249 }
1251 /*
1252  * get device's mount point from mnttab
1253  */
1254 char *
1255 rmm_get_mnttab_mount_point(const char *special)
1256 {
1257     char *mount_point = NULL;
1258     FILE *f;
1259     struct mnttab mnt;
1260     struct mnttab mpref = { NULL, NULL, NULL, NULL, NULL };
1262     if ((f = fopen(MNTTAB, "r")) != NULL) {
1263         mpref.mnt_special = (char *)special;
1264         if (getmntany(f, &mnt, &mpref) == 0) {
1265             mount_point = strdup(mnt.mnt_mountp);
1266         }
1267         fclose(f);
1268     }
1270     return (mount_point);
1271 }
1274 /*
1275  * get human readable string from error values
1276  */
1277 const char *
1278 rmm_strerror(DBusError *dbus_error, int rmm_error)
1279 {
1280     const char *str;
1282     if ((dbus_error != NULL) && dbus_error_is_set(dbus_error)) {
1283         str = dbus_error->message;
1284     } else {
1285         switch (rmm_error) {
1286             case RMM_EOK:
1287                 str = gettext("success");
1288                 break;
1289             case RMM_EDBUS_CONNECT:
1290                 str = gettext("cannot connect to D-Bus");
1291                 break;
1292             case RMM_EHAL_CONNECT:
1293                 str = gettext("cannot connect to HAL");
1294                 break;
1295             default:
1296                 str = gettext("undefined error");
1297                 break;
1298         }
1299     }
1301     return (str);
1302 }

```

```

1304 void
1305 rmm_dbus_error_free(DBusError *error)
1306 {
1307     if (error != NULL && dbus_error_is_set(error)) {
1308         dbus_error_free(error);
1309     }
1310 }

1312 static int
1313 rmm_vold_isbadchar(int c)
1314 {
1315     int    ret_val = 0;

1318     switch (c) {
1319     case '/':
1320     case ';':
1321     case '|':
1322         ret_val = 1;
1323         break;
1324     default:
1325         if (iscntrl(c) || isspace(c)) {
1326             ret_val = 1;
1327         }
1328     }

1330     return (ret_val);
1331 }

1333 char *
1334 rmm_vold_convert_volume_label(const char *name, size_t len)
1335 {
1336     char    buf[MAXNAMELEN+1];
1337     char    *s = buf;
1338     int     i;

1340     if (len > MAXNAMELEN) {
1341         len = MAXNAMELEN;
1342     }

1344     for (i = 0; i < len; i++) {
1345         if (name[i] == '\0') {
1346             break;
1347         }
1348         if (isgraph((int)name[i])) {
1349             if (isupper((int)name[i])) {
1350                 *s++ = tolower((int)name[i]);
1351             } else if (rmm_vold_isbadchar((int)name[i])) {
1352                 *s++ = '_';
1353             } else {
1354                 *s++ = name[i];
1355             }
1356         }
1357     }
1358     *s = '\0';
1359     s = strdup(buf);

1361     return (s);
1362 }

1364 /*
1365  * swiped from mkdir.c
1366  */
1367 int
1368 makepath(char *dir, mode_t mode)
1369 {

```

```

1370     int     err;
1371     char    *slash;

1374     if ((mkdir(dir, mode) == 0) || (errno == EEXIST)) {
1375         return (0);
1376     }
1377     if (errno != ENOENT) {
1378         return (-1);
1379     }
1380     if ((slash = strrchr(dir, '/')) == NULL) {
1381         return (-1);
1382     }
1383     *slash = '\0';
1384     err = makepath(dir, mode);
1385     *slash++ = '/';

1387     if (err || (*slash == '\0')) {
1388         return (err);
1389     }

1391     return (mkdir(dir, mode));
1392 }

1395 void
1396 dprintf(const char *fmt, ...)
1397 {
1399     va_list ap;
1400     const char *p;
1401     char    msg[BUFSIZ];
1402     char    *errmsg = strerror(errno);
1403     char    *s;

1405     if (rmm_debug == 0) {
1406         return;
1407     }

1409     (void) memset(msg, 0, BUFSIZ);

1411     /* scan for %m and replace with errno msg */
1412     s = &msg[strlen(msg)];
1413     p = fmt;

1415     while (*p != '\0') {
1416         if ((*p == '%') && (*(p+1) == 'm')) {
1417             (void) strcat(s, errmsg);
1418             p += 2;
1419             s += strlen(errmsg);
1420             continue;
1421         }
1422         *s++ = *p++;
1423     }
1424     *s = '\0';    /* don't forget the null byte */

1426     va_start(ap, fmt);
1427     (void) vfprintf(stderr, msg, ap);
1428     va_end(ap);
1429 }

```

new/usr/src/cmd/rmvolmgr/vold.c

1

```
*****
29524 Sun May 4 18:28:37 2014
new/usr/src/cmd/rmvolmgr/vold.c
4833 Remove volrmmount
4845 rm(u)mount don't always print mount/unmount errors
4846 HAL partition names don't match real partition names
Reviewed by: Dan McDonald <danmcd@omniti.com>
Reviewed by: Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 *
21 */
22 * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 *
25 * Copyright 2014 Andrew Stormont
26 #endif /* ! codereview */
27 */
28
29 /*
30 * Vold compatibility for rmvolmgr: emulate old commands as well as
31 * action_filemgr.so to notify legacy apps via /tmp/.removable pipes.
32 * A lot of this code is copied verbatim from vold sources.
33 *
34 * Here's the original description of action_filemgr.so:
35 *
36 * action_filemgr.so - filemgr interface routines for rmmount
37 *
38 * This shared object allows rmmount to communicate with filemgr.
39 * This is done by communicating over a named pipe that filemgr
40 * creates in directory NOTIFY_DIR. The name of the pipe must
41 * begin with NOTIFY_NAME. This source file contains #define
42 * compiler directives set the values of NOTIFY_DIR and NOTIFY_NAME.
43 *
44 * After a partition on a medium has been mounted as a result of
45 * either insertion or remounting of the medium, the action()
46 * method creates a file named with the symbolic name of the
47 * device in which the medium is inserted and the partition name
48 * (e.g. "jaz0-s2") in NOTIFY_DIR. The file consists of one text
49 * line containing a string naming the mount point of the partition,
50 * a string giving the raw device path to the partition, and a
51 * string naming the file system type on the partition. The action()
52 * method then sends a single character ('i' for insertion, 'r' for
53 * remounting) through the named pipe NOTIFY_NAME to tell filemgr to
54 * look for new files in NOTIFY_DIR.
55 */
```

new/usr/src/cmd/rmvolmgr/vold.c

2

```
56 * If a medium containing no mountable partitions is inserted
57 * or remounted in a device, the action() method creates a file
58 * named with the symbolic name of the device in NOTIFY_DIR.
59 * The file consists of one text line containing a string
60 * giving the symbolic name of the device and a string naming
61 * the reason that the medium couldn't be mounted. The action
62 * method then sends either an 'i' or an 'r' through the named
63 * pipe to tell filemgr to look for new files in NOTIFY_DIR.
64 *
65 * When a medium is ejected or unmounted, the action() method
66 * removes the files that were created in NOTIFY_DIR when the medium
67 * was inserted or remounted and sends a single character ('e' for
68 * ejection, 'u' for unmounting) through the named pipe.
69 *
70 * The following environment variables must be set before calling action():
71 *
72 *     VOLUME_ACTION      action that occurred (e.g. "insert", "eject")
73 *     VOLUME_SYMDEV      symbolic name (e.g. "cdrom0", "floppy1")
74 *     VOLUME_NAME        volume name (e.g. "unnamed_cdrom", "s2")
75 */
76
77 #include <stdio.h>
78 #include <stdlib.h>
79 #include <unistd.h>
80 #include <fcntl.h>
81 #include <string.h>
82 #include <strings.h>
83 #include <dirent.h>
84 #include <signal.h>
85 #include <errno.h>
86 #include <libintl.h>
87 #include <zone.h>
88 #include <pwd.h>
89 #include <sys/types.h>
90 #include <sys/stat.h>
91 #include <sys/dkio.h>
92 #include <sys/cdio.h>
93 #include <sys/vtoci.h>
94 #include <sys/param.h>
95 #include <sys/wait.h>
96 #include <libcontract.h>
97 #include <sys/contract/process.h>
98 #include <sys/ctfs.h>
99 #include <tsol/label.h>
100
101
102 #include "vold.h"
103 #include "rmm_common.h"
104
105 int          rmm_debug = 0;
106 boolean_t   rmm_vold_actions_enabled = B_FALSE;
107 boolean_t   rmm_vold_mountpoints_enabled = B_FALSE;
108
109 static char *prog_name = NULL;
110 static pid_t prog_pid = 0;
111 static int   system_labeled = 0;
112 static uid_t mnt_uid = (uid_t)-1;
113 static gid_t mnt_gid = (gid_t)-1;
114 static zoneid_t mnt_zoneid = -1;
115 static char   mnt_zoneroot[MAXPATHLEN];
116 static char   mnt_userdir[MAXPATHLEN];
117
118 /*
119 * Private attribute types and attributes.
120 */
121 static const char notify_characters[] = {
```

```
122     'e',
123     'i',
124     'r',
125     'u'
126 };

128 static const char *result_strings[] = {
129     "FALSE",
130     "TRUE"
131 };

133 #define NOTIFY_DIR      "/tmp/.removable"      /* dir where filemgr looks */
134 #define NOTIFY_NAME    "notify"              /* named pipe to talk over */

 24 static void      volrmount_usage();
136 static void      volcheck_usage();
137 static int       vold_action(struct action_arg *aap);
138 static void      vold_update_mountpoints(struct action_arg *aap);
139 static char      *not_mountable(struct action_arg *aa);
140 static int       create_one_notify_file(char *fstype,
141                                         char *mount_point,
142                                         char *notify_file,
143                                         char *raw_partitionp,
144                                         char *reason,
145                                         char *symdev);
146 static int       create_notify_files(struct action_arg **aa);
147 static boolean_t notify_clients(action_t action, int do_notify);
148 static void      popdir(int fd);
149 static int       pushdir(const char *dir);
150 static boolean_t remove_notify_files(struct action_arg **aa);

152 /*
153  * should be called once from main()
154  */
155 /* ARGSUSED */
156 void
157 vold_init(int argc, char **argv)
158 {
159     system_labeled = is_system_labeled();
160 }
  unchanged_portion_omitted_
```

new/usr/src/cmd/rmvolmgr/vold.h

1

```
*****
1974 Sun May 4 18:28:37 2014
new/usr/src/cmd/rmvolmgr/vold.h
4833 Remove volrmmount
4845 rm(u)mount don't always print mount/unmount errors
4846 HAL partition names don't match real partition names
Reviewed by: Dan McDonald <danmcd@omniti.com>
Reviewed by: Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
*****
```

```
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 *
21 *
20 */
21 /*
22 * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 *
25 * Copyright 2014 Andrew Stormont.
26 #endif /* ! codereview */
27 */

29 #ifndef _VOLD_H
30 #define _VOLD_H

24 #pragma ident "%Z%M% %I% %E% SMI"

32 #ifdef __cplusplus
33 extern "C" {
34 #endif

36 #include <libhal.h>

38 typedef enum {
39     EJECT,
40     INSERT,
41     REMOUNT,
42     UNMOUNT,
43     CLEAR_MOUNTS,
44     CLOSETRAY
45 } action_t;
_____ unchanged_portion_omitted _____

59 extern int rmm_debug;
60 extern boolean_t rmm_vold_actions_enabled;
61 extern boolean_t rmm_vold_mountpoints_enabled;

63 void vold_init(int argc, char **argv);
64 int vold_postprocess(LibHalContext *hal_ctx, const char *udi,
```

new/usr/src/cmd/rmvolmgr/vold.h

2

```
65     struct action_arg *aap);
66 int vold_rmmount(int argc, char **argv);
61 int volrmmount(int argc, char **argv);
67 int volcheck(int argc, char **argv);

69 #ifdef __cplusplus
70 }
_____ unchanged_portion_omitted _____
```



```

*****
13118 Sun May 4 18:28:37 2014
new/usr/src/man/man1/Makefile
4833 Remove volrmmount
4845 rm(u)mount don't always print mount/unmount errors
4846 HAL partition names don't match real partition names
Reviewed by: Dan McDonald <danmcd@omniti.com>
Reviewed by: Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
*****
1 #
2 # This file and its contents are supplied under the terms of the
3 # Common Development and Distribution License ("CDDL"), version 1.0.
4 # You may only use this file in accordance with the terms of version
5 # 1.0 of the CDDL.
6 #
7 # A full copy of the text of the CDDL should have accompanied this
8 # source. A copy of the CDDL is also available via the Internet
9 # at http://www.illumos.org/license/CDDL.
10 #
11 #
12 #
13 # Copyright 2011, Richard Lowe
14 # Copyright 2013 Nexenta Systems, Inc. All rights reserved.
15 # Copyright 2014 Andrew Stormont.
16 #endif /* ! codereview */
17 #
18 #
19 include $(SRC)/Makefile.master
20 #
21 MANSECT= 1
22 #
23 MANFILES= acctcom.1 \
24 adb.1 \
25 addbib.1 \
26 alias.1 \
27 allocate.1 \
28 amt.1 \
29 appcert.1 \
30 apptrace.1 \
31 apropos.1 \
32 ar.1 \
33 arch.1 \
34 asa.1 \
35 at.1 \
36 atq.1 \
37 atrm.1 \
38 audioconvert.1 \
39 audiocvt.1 \
40 audioplay.1 \
41 audiorecord.1 \
42 audiotest.1 \
43 auths.1 \
44 awk.1 \
45 banner.1 \
46 basename.1 \
47 bc.1 \
48 bdiff.1 \
49 bfs.1 \
50 break.1 \
51 builtin.1 \
52 cal.1 \
53 calendar.1 \
54 cancel.1 \
55 cat.1 \
56 cd.1 \
57 cdrw.1 \

```

```

58 checknr.1 \
59 chgrp.1 \
60 chkey.1 \
61 chmod.1 \
62 chown.1 \
63 ckdate.1 \
64 ckgid.1 \
65 ckint.1 \
66 ckitem.1 \
67 ckkeywd.1 \
68 ckpath.1 \
69 ckrange.1 \
70 ckstr.1 \
71 cksum.1 \
72 cktime.1 \
73 ckuid.1 \
74 ckyorn.1 \
75 clear.1 \
76 cmp.1 \
77 col.1 \
78 comm.1 \
79 command.1 \
80 compress.1 \
81 cp.1 \
82 cpio.1 \
83 cputrack.1 \
84 crle.1 \
85 crontab.1 \
86 crypt.1 \
87 csh.1 \
88 csplit.1 \
89 ctags.1 \
90 ctrun.1 \
91 ctstat.1 \
92 ctwatch.1 \
93 cut.1 \
94 date.1 \
95 dc.1 \
96 deallocate.1 \
97 deroff.1 \
98 dhcpinfo.1 \
99 diff.1 \
100 diff3.1 \
101 diffmk.1 \
102 digest.1 \
103 dircmp.1 \
104 dis.1 \
105 disown.1 \
106 dispuid.1 \
107 dispuid.1 \
108 dos2unix.1 \
109 download.1 \
110 dpost.1 \
111 du.1 \
112 dump.1 \
113 dumpcs.1 \
114 echo.1 \
115 ed.1 \
116 egrep.1 \
117 eject.1 \
118 elfdump.1 \
119 elfedit.1 \
120 elfsign.1 \
121 elfwrap.1 \
122 enable.1 \
123 encrypt.1 \

```

```

124      enhance.1      \
125      env.1          \
126      eqn.1          \
127      exec.1         \
128      exit.1         \
129      expand.1       \
130      expr.1         \
131      exstr.1        \
132      factor.1       \
133      fdformat.1     \
134      fgrep.1        \
135      file.1         \
136      filebench.1   \
137      filesync.1    \
138      find.1         \
139      finger.1      \
140      fmt.1          \
141      fmtmsg.1      \
142      fold.1         \
143      ftp.1          \
144      ftpcount.1    \
145      ftpwho.1      \
146      gcore.1       \
147      gencat.1      \
148      genmsg.1      \
149      getconf.1     \
150      getfacl.1     \
151      getlabel.1    \
152      getopt.1      \
153      getoptcv.1    \
154      getopts.1     \
155      gettext.1     \
156      gettxt.1      \
157      getzonepath.1 \
158      glob.1         \
159      gprof.1       \
160      grep.1         \
161      groups.1      \
162      hash.1         \
163      head.1         \
164      history.1     \
165      hostid.1      \
166      hostname.1    \
167      iconv.1       \
168      indxbib.1     \
169      Intro.1       \
170      ipcrm.1       \
171      ipcs.1        \
172      isainfo.1     \
173      isalist.1     \
174      jobs.1        \
175      join.1        \
176      kbd.1         \
177      kdestroy.1    \
178      keylogin.1    \
179      keylogout.1   \
180      kill.1        \
181      kinit.1       \
182      klist.1       \
183      kmdb.1        \
184      kmfcfg.1      \
185      kpasswd.1     \
186      krb5-config.1 \
187      ksh93.1       \
188      ktutil.1      \
189      lari.1        \

```

```

190      last.1         \
191      lastcomm.1    \
192      ld.1          \
193      ldap.1        \
194      ldapdelete.1  \
195      ldaplist.1    \
196      ldapmodify.1  \
197      ldapmodrdn.1  \
198      ldapsearch.1  \
199      ldd.1         \
200      ld.so.1.1     \
201      let.1         \
202      lex.1         \
203      lgrpinfo.1    \
204      limit.1       \
205      line.1        \
206      list_devices.1 \
207      listusers.1   \
208      ln.1          \
209      loadkeys.1    \
210      locale.1      \
211      localedef.1   \
212      logger.1      \
213      login.1       \
214      logname.1     \
215      logout.1      \
216      look.1        \
217      lookbib.1     \
218      lorder.1     \
219      lp.1          \
220      lpstat.1      \
221      ls.1          \
222      m4.1          \
223      mac.1         \
224      mach.1        \
225      machid.1     \
226      madv.so.1.1  \
227      mail.1        \
228      mailcompat.1  \
229      mailq.1       \
230      mailstats.1  \
231      mailx.1       \
232      makekey.1     \
233      man.1         \
234      mconnect.1    \
235      mcs.1         \
236      mdb.1         \
237      msg.1         \
238      mkdir.1       \
239      mkmsgs.1      \
240      mktemp.1      \
241      moe.1         \
242      more.1        \
243      mpss.so.1.1  \
244      msgcc.1       \
245      msgcpp.1      \
246      msgcvt.1      \
247      msgfmt.1      \
248      msggen.1      \
249      msgget.1      \
250      mt.1          \
251      mv.1          \
252      nawk.1        \
253      nc.1          \
254      nca.1         \
255      ncab2clf.1    \

```

```

256 ncakmod.1 \
257 newform.1 \
258 newgrp.1 \
259 news.1 \
260 newtask.1 \
261 nice.1 \
262 nl.1 \
263 nm.1 \
264 nohup.1 \
265 nroff.1 \
266 od.1 \
267 on.1 \
268 optisa.1 \
269 pack.1 \
270 pagesize.1 \
271 pargs.1 \
272 passwd.1 \
273 paste.1 \
274 pathchk.1 \
275 pax.1 \
276 pfexec.1 \
277 pg.1 \
278 pgrep.1 \
279 pkginfo.1 \
280 pkgmk.1 \
281 pkgparam.1 \
282 pkgproto.1 \
283 pkgtrans.1 \
284 pktool.1 \
285 plabel.1 \
286 plgrp.1 \
287 plimit.1 \
288 pmadvise.1 \
289 pmap.1 \
290 postio.1 \
291 postprint.1 \
292 postreverse.1 \
293 ppgsz.1 \
294 ppriv.1 \
295 pr.1 \
296 praliases.1 \
297 prctl.1 \
298 preap.1 \
299 prex.1 \
300 print.1 \
301 printf.1 \
302 priocntl.1 \
303 proc.1 \
304 prof.1 \
305 profiles.1 \
306 projects.1 \
307 ps.1 \
308 ptree.1 \
309 pvs.1 \
310 pwd.1 \
311 ranlib.1 \
312 rcapstat.1 \
313 rcp.1 \
314 rdist.1 \
315 read.1 \
316 readonly.1 \
317 refer.1 \
318 regcmp.1 \
319 renice.1 \
320 rev.1 \
321 rlogin.1 \

```

```

322 rm.1 \
323 rmformat.1 \
324 rmmount.1 \
325 roffbib.1 \
326 roles.1 \
327 rpcgen.1 \
328 rsh.1 \
329 runat.1 \
330 rup.1 \
331 ruptime.1 \
332 rusers.1 \
333 rwho.1 \
334 sar.1 \
335 scp.1 \
336 script.1 \
337 sdiff.1 \
338 sed.1 \
339 set.1 \
340 setfacl.1 \
341 setlabel.1 \
342 setpgpr.1 \
343 sftp.1 \
344 shcomp.1 \
345 shell_builtins.1 \
346 shift.1 \
347 size.1 \
348 sleep.1 \
349 smbutil.1 \
350 soelim.1 \
351 sort.1 \
352 sortbib.1 \
353 sotruss.1 \
354 spell.1 \
355 split.1 \
356 srchtxt.1 \
357 ssh.1 \
358 ssh-add.1 \
359 ssh-agent.1 \
360 ssh-http-proxy-connect.1 \
361 ssh-keygen.1 \
362 ssh-keyscan.1 \
363 ssh-socks5-proxy-connect.1 \
364 strchg.1 \
365 strings.1 \
366 strip.1 \
367 stty.1 \
368 sum.1 \
369 suspend.1 \
370 svcprop.1 \
371 svcs.1 \
372 symorder.1 \
373 sys-suspend.1 \
374 tabs.1 \
375 tail.1 \
376 talk.1 \
377 tar.1 \
378 tbl.1 \
379 tcopy.1 \
380 tee.1 \
381 telnet.1 \
382 test.1 \
383 tftp.1 \
384 time.1 \
385 times.1 \
386 timex.1 \
387 tip.1 \

```

```

388          tnfdump.1          \|
389          tnfxtract.1       \|
390          touch.1           \|
391          tput.1            \|
392          tr.1              \|
393          trap.1            \|
394          troff.1           \|
395          true.1            \|
396          truss.1           \|
397          tsort.1           \|
398          tty.1             \|
399          type.1            \|
400          typeset.1         \|
401          ul.1              \|
402          umask.1           \|
403          uname.1           \|
404          unifdef.1         \|
405          uniq.1            \|
406          units.1           \|
407          unix2dos.1        \|
408          uptime.1          \|
409          vacation.1        \|
410          vgrind.1          \|
411          volcheck.1        \|
15          volrmount.1      \|
412          w.1              \|
413          wait.1            \|
414          wc.1              \|
415          whatis.1          \|
416          which.1           \|
417          who.1             \|
418          whocalls.1        \|
419          whois.1           \|
420          write.1           \|
421          xargs.1           \|
422          xgettext.1        \|
423          xstr.1            \|
424          yacc.1            \|
425          yes.1             \|
426          ypcat.1           \|
427          ypmatch.1         \|
428          yppasswd.1        \|
429          ypwhich.1         \|
430          zlogin.1          \|
431          zonename.1        \|

433 MANLINKS= batch.1        \|
434             bg.1           \|
435             case.1         \|
436             chdir.1        \|
437             checkeg.1      \|
438             continue.1     \|
439             decrypt.1      \|
440             dirname.1       \|
441             dirs.1          \|
442             disable.1       \|
443             dumpkeys.1     \|
444             edit.1          \|
445             errange.1       \|
446             errdate.1       \|
447             errgid.1        \|
448             errint.1        \|
449             erritem.1       \|
450             errpath.1       \|
451             errstr.1        \|
452             errtime.1       \|

```

```

453          erruid.1          \|
454          erryorn.1         \|
455          eval.1            \|
456          export.1          \|
457          false.1           \|
458          fc.1              \|
459          fg.1              \|
460          for.1             \|
461          foreach.1         \|
462          function.1        \|
463          goto.1            \|
464          hashcheck.1       \|
465          hashmake.1        \|
466          hashstat.1        \|
467          helpdate.1        \|
468          helpgid.1         \|
469          helpint.1         \|
470          helpitem.1        \|
471          helppath.1        \|
472          helprange.1       \|
473          helpstr.1         \|
474          helptime.1        \|
475          helpuid.1         \|
476          helpyorn.1        \|
477          hist.1            \|
478          i286.1            \|
479          i386.1            \|
480          i486.1            \|
481          i860.1            \|
482          iAPX286.1         \|
483          if.1              \|
484          intro.1           \|
485          jsh.1             \|
486          ksh.1             \|
487          ldapadd.1         \|
488          neqn.1            \|
489          notify.1          \|
490          onintr.1          \|
491          page.1            \|
492          pcat.1            \|
493          pcred.1           \|
494          pdpl1.1           \|
495          pfcsh.1           \|
496          pfiles.1          \|
497          pfksh.1           \|
498          pflags.1          \|
499          pfsh.1            \|
500          pkill.1           \|
501          pldd.1            \|
502          popd.1            \|
503          prun.1            \|
504          psig.1            \|
505          pstack.1          \|
506          pstop.1           \|
507          ptime.1           \|
508          pushd.1           \|
509          pwait.1           \|
510          pwdx.1            \|
511          red.1             \|
512          rehash.1          \|
513          remote_shell.1    \|
514          remsh.1           \|
515          repeat.1          \|
516          return.1          \|
517          rksh.1            \|
518          rksh93.1          \|

```

```

519      rmail.1      \|
520      rmdir.1     \|
521      rmumount.1  \|
522      select.1   \|
523      setenv.1   \|
524      settime.1  \|
525      sh.1       \|
526      snca.1    \|
527      source.1  \|
528      sparc.1   \|
529      spellin.1 \|
530      stop.1    \|
531      strconf.1 \|
532      sun.1     \|
533      switch.1  \|
534      u370.1   \|
535      u3b.1    \|
536      u3b15.1  \|
537      u3b2.1   \|
538      u3b5.1   \|
539      ulimit.1  \|
540      unalias.1 \|
541      uncompress.1 \|
542      unexpand.1 \|
543      unhash.1  \|
544      unlimit.1 \|
545      unpack.1  \|
546      unset.1   \|
547      unsetenv.1 \|
548      until.1   \|
549      valdate.1 \|
550      valgid.1  \|
551      valint.1  \|
552      valpath.1 \|
553      valrange.1 \|
554      valstr.1  \|
555      valtime.1 \|
556      valuid.1  \|
557      valyorn.1 \|
558      vax.1     \|
559      vedit.1   \|
560      whence.1  \|
561      while.1   \|
562      zcat.1    \|

564 intro.1      := LINKSRC = Intro.1

566 unalias.1    := LINKSRC = alias.1

568 batch.1      := LINKSRC = at.1

570 dirname.1    := LINKSRC = basename.1

572 continue.1  := LINKSRC = break.1

574 chdir.1      := LINKSRC = cd.1
575 dirs.1       := LINKSRC = cd.1
576 popd.1       := LINKSRC = cd.1
577 pushd.1      := LINKSRC = cd.1

579 errdate.1    := LINKSRC = ckdate.1
580 helpdate.1   := LINKSRC = ckdate.1
581 valdate.1    := LINKSRC = ckdate.1

583 errgid.1     := LINKSRC = ckgid.1
584 helpgid.1    := LINKSRC = ckgid.1

```

```

585 valgid.1     := LINKSRC = ckgid.1

587 errint.1     := LINKSRC = ckint.1
588 helpint.1    := LINKSRC = ckint.1
589 valint.1     := LINKSRC = ckint.1

591 erritem.1    := LINKSRC = ckitem.1
592 helpitem.1   := LINKSRC = ckitem.1

594 errpath.1    := LINKSRC = ckpath.1
595 helppath.1   := LINKSRC = ckpath.1
596 valpath.1    := LINKSRC = ckpath.1

598 errrange.1   := LINKSRC = ckrange.1
599 helprange.1  := LINKSRC = ckrange.1
600 valrange.1   := LINKSRC = ckrange.1

602 errstr.1     := LINKSRC = ckstr.1
603 helpstr.1    := LINKSRC = ckstr.1
604 valstr.1     := LINKSRC = ckstr.1

606 errtime.1    := LINKSRC = cktime.1
607 helptime.1  := LINKSRC = cktime.1
608 valtime.1    := LINKSRC = cktime.1

610 erruid.1     := LINKSRC = ckuid.1
611 helpuid.1   := LINKSRC = ckuid.1
612 valuid.1     := LINKSRC = ckuid.1

614 erryorn.1    := LINKSRC = ckyorn.1
615 helpyorn.1   := LINKSRC = ckyorn.1
616 valyorn.1    := LINKSRC = ckyorn.1

618 uncompress.1 := LINKSRC = compress.1
619 zcat.1       := LINKSRC = compress.1

621 red.1        := LINKSRC = ed.1

623 disable.1   := LINKSRC = enable.1

625 decrypt.1   := LINKSRC = encrypt.1

627 checkeq.1   := LINKSRC = eqn.1
628 neqn.1      := LINKSRC = eqn.1

630 eval.1       := LINKSRC = exec.1
631 source.1     := LINKSRC = exec.1

633 goto.1      := LINKSRC = exit.1
634 return.1    := LINKSRC = exit.1

636 unexpand.1  := LINKSRC = expand.1

638 hashstat.1  := LINKSRC = hash.1
639 rehash.1    := LINKSRC = hash.1
640 unhash.1    := LINKSRC = hash.1

642 fc.1        := LINKSRC = history.1
643 hist.1      := LINKSRC = history.1

645 bg.1        := LINKSRC = jobs.1
646 fg.1        := LINKSRC = jobs.1
647 notify.1    := LINKSRC = jobs.1
648 stop.1      := LINKSRC = jobs.1

650 jsh.1       := LINKSRC = ksh93.1

```

new/usr/src/man/man1/Makefile

11

```

651 ksh.1      := LINKSRC = ksh93.1
652 rksh.1     := LINKSRC = ksh93.1
653 rksh93.1   := LINKSRC = ksh93.1
654 sh.1       := LINKSRC = ksh93.1

656 ldapadd.1  := LINKSRC = ldapmodify.1

658 ulimit.1   := LINKSRC = limit.1
659 unlimit.1  := LINKSRC = limit.1

661 dumpkeys.1 := LINKSRC = loadkeys.1

663 i286.1     := LINKSRC = machid.1
664 i386.1     := LINKSRC = machid.1
665 i486.1     := LINKSRC = machid.1
666 i860.1     := LINKSRC = machid.1
667 iAPX286.1  := LINKSRC = machid.1
668 pdp11.1    := LINKSRC = machid.1
669 sparc.1    := LINKSRC = machid.1
670 sun.1      := LINKSRC = machid.1
671 u370.1     := LINKSRC = machid.1
672 u3b.1      := LINKSRC = machid.1
673 u3b15.1   := LINKSRC = machid.1
674 u3b2.1    := LINKSRC = machid.1
675 u3b5.1    := LINKSRC = machid.1
676 vax.1     := LINKSRC = machid.1

678 rmail.1    := LINKSRC = mail.1

680 page.1     := LINKSRC = more.1

682 snca.1     := LINKSRC = nca.1

684 pcat.1     := LINKSRC = pack.1
685 unpack.1   := LINKSRC = pack.1

687 pfcsh.1    := LINKSRC = pfexec.1
688 pfksh.1    := LINKSRC = pfexec.1
689 pfsh.1     := LINKSRC = pfexec.1

691 pkill.1    := LINKSRC = pgrep.1

693 pcred.1    := LINKSRC = proc.1
694 pfiles.1   := LINKSRC = proc.1
695 pflags.1   := LINKSRC = proc.1
696 pldd.1     := LINKSRC = proc.1
697 prun.1     := LINKSRC = proc.1
698 psig.1     := LINKSRC = proc.1
699 pstack.1   := LINKSRC = proc.1
700 pstop.1    := LINKSRC = proc.1
701 ptime.1    := LINKSRC = proc.1
702 pwait.1    := LINKSRC = proc.1
703 pwdx.1     := LINKSRC = proc.1

705 rmdir.1    := LINKSRC = rm.1

707 rmumount.1 := LINKSRC = rmmount.1

709 remote_shell.1 := LINKSRC = rsh.1
710 remsh.1     := LINKSRC = rsh.1

712 export.1   := LINKSRC = set.1
713 setenv.1   := LINKSRC = set.1
714 unset.1    := LINKSRC = set.1
715 unsetenv.1 := LINKSRC = set.1

```

new/usr/src/man/man1/Makefile

12

```

717 case.1     := LINKSRC = shell_builtins.1
718 for.1       := LINKSRC = shell_builtins.1
719 foreach.1   := LINKSRC = shell_builtins.1
720 function.1  := LINKSRC = shell_builtins.1
721 if.1        := LINKSRC = shell_builtins.1
722 repeat.1    := LINKSRC = shell_builtins.1
723 select.1    := LINKSRC = shell_builtins.1
724 switch.1    := LINKSRC = shell_builtins.1
725 until.1    := LINKSRC = shell_builtins.1
726 while.1    := LINKSRC = shell_builtins.1

728 hashcheck.1 := LINKSRC = spell.1
729 hashmake.1  := LINKSRC = spell.1
730 spellin.1   := LINKSRC = spell.1

732 strconf.1  := LINKSRC = strchg.1

734 settime.1  := LINKSRC = touch.1

736 onintr.1   := LINKSRC = trap.1

738 false.1    := LINKSRC = true.1

740 whence.1   := LINKSRC = typeset.1

742 # Links to usr/has/man

744 edit.1     := LINKSRC = ../../../has/man/man1has/edit.lhas

746 vedit.1    := LINKSRC = ../../../has/man/man1has/vi.lhas

748 .KEEP_STATE:

750 include    $(SRC)/man/Makefile.man

752 install:   $(ROOTMANFILES) $(ROOTMANLINKS)

```

```

*****
11823 Sun May 4 18:28:37 2014
new/usr/src/man/man1/fdformat.1
4833 Remove volrmmount
4845 rm(u)mount don't always print mount/unmount errors
4846 HAL partition names don't match real partition names
Reviewed by: Dan McDonald <danmcd@omniti.com>
Reviewed by: Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
*****
1 \" te
2 .\\ Copyright 2001, Sun Microsystems, Inc All Rights Reserved
3 .\\ Copyright 2014 Andrew Stormont.
4 #endif /* ! codereview */
5 .\\ The contents of this file are subject to the terms of the Common Development
6 .\\ You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
7 .\\ When distributing Covered Code, include this CDDL HEADER in each file and in
8 .TH FDFORMAT 1 \"May 2, 2014\"
9 .SH NAME
10 fdformat \\- format floppy diskette or PCMCIA memory card
11 .SH SYNOPSIS
12 .LP
13 .nf
14 \\fbfdformat\\fR [\\fB-dDeFhLmMUgvx\\fR] [\\fB-b\\fR \\fIlabel\\fR] [\\fB-B\\fR \\fIfile
15 [\\fB-t\\fR \\fIdostype\\fR] [\\fIdevname\\fR]
16 .fi

18 .SH DESCRIPTION
19 .sp
20 .LP
21 The \\fbfdformat\\fR utility has been superseded by \\fBrmformat\\fR(1), which
22 provides most but not all of \\fbfdformat\\fR's functionality.
23 .sp
24 .LP
25 \\fbfdformat\\fR is used to format diskettes and \\fbPCMCIA\\fR memory cards. All
26 new blank diskettes or \\fbPCMCIA\\fR memory cards must be formatted before they
27 can be used.
28 .sp
29 .LP
30 \\fbfdformat\\fR formats and verifies the media and indicates whether any bad
31 sectors were encountered. All existing data on the diskette or \\fbPCMCIA\\fR
32 memory card, if any, is destroyed by formatting. If no device name is given,
33 \\fbfdformat\\fR uses the diskette as a default.
34 .sp
35 .LP
36 By default, \\fbfdformat\\fR uses the configured capacity of the drive to format
37 the diskette. A \\fB3.5\\fR inch high-density drive uses diskettes with a
38 formatted capacity of \\fB1.44MB\\fR. A \\fB5.25\\fR inch high-density drive uses
39 diskettes with a formatted capacity of \\fB1.2MB\\fR. In either case, a density
40 option does not have to be specified to \\fbfdformat\\fR. However, a density
41 option must be specified when using a diskette with a lower capacity than the
42 drive's default. Use the \\fB-H\\fR option to format high-density diskettes
43 (\\fB1.44MB\\fR capacity) in an extra-high-density (\\fBED\\fR) drive. Use the
44 \\fB-D\\fR option, the \\fB-l\\fR option, or the \\fB-L\\fR option to format double-
45 density (or low-density) diskettes (\\fB720KB\\fR capacity) in an \\fBHd\\fR or
46 \\fBED\\fR drive. To format medium-density diskettes (\\fB1.2MB\\fR capacity), use
47 the \\fB-M\\fR option with \\fB-t\\fR \\fBnec\\fR (this is the same as using the
48 \\fB-m\\fR option with \\fB-t\\fR \\fBnec\\fR).
49 .sp
50 .LP
51 Extended density uses double-sided, extended-density or extra-high-density
52 (\\fBDS\\fR/\\fBED\\fR) diskettes. Medium and high densities use the same media:
53 double-sided, high-density (\\fBDS\\fR/\\fBHD\\fR) diskettes. Double (low) density
54 uses double-sided, double-density (\\fBDS\\fR/\\fBDD\\fR)
55 .sp
56 .LP

```

```

57 \\fBD\\fR) diskettes. Substituting diskettes of one density for diskettes of
58 either a higher or lower density generally does not work. Data integrity cannot
59 be assured whenever a diskette is formatted to a capacity not matching its
60 density.
61 .sp
62 .LP
63 A \\fbPCMCIA\\fR memory card with densities from \\fB512KB\\fR to \\fB64MB\\fR may be
64 formatted.
65 .sp
66 .LP
67 \\fbfdformat\\fR writes new identification and data fields for each sector on all
68 tracks unless the \\fB-x\\fR option is specified. For diskettes, each sector is
69 verified if the \\fB-v\\fR option is specified.
70 .sp
71 .LP
72 After formatting and verifying, \\fbfdformat\\fR writes an operating-system label
73 on block \\fB0\\fR. Use the \\fB-t\\fR \\fBdos\\fR option (same as the \\fB-d\\fR
74 option) to put an \\fBMS-DOS\\fR file system on the diskette or \\fbPCMCIA\\fR
75 memory card after the format is done. Use the \\fB-t\\fR \\fBnec\\fR option with
76 the \\fB-M\\fR option (same as the \\fB-m\\fR option) to put an \\fBNEC-DOS\\fR file
77 system on a diskette. Otherwise, \\fbfdformat\\fR writes a \\fBSunOS\\fR label in
78 block \\fB0\\fR.
79 .SH OPTIONS
80 .sp
81 .LP
82 The following options are supported:
83 .sp
84 .ne 2
85 .na
86 \\fB\\fB-b\\fR \\fIlabel\\fR\\fR
87 .ad
88 .RS 15n
89 Labels the media with volume label. A SunOS volume label is restricted to 8
90 characters. A \\fBDOS\\fR volume label is restricted to 11 upper-case characters.
91 .RE

93 .sp
94 .ne 2
95 .na
96 \\fB\\fB-B\\fR \\fIfilename\\fR\\fR
97 .ad
98 .RS 15n
99 Installs special boot loader in filename on an \\fBMS-DOS\\fR diskette. This
100 option is only meaningful when the \\fB-d\\fR option (or \\fB-t\\fR \\fBdos\\fR) is
101 also specified.
102 .RE

104 .sp
105 .ne 2
106 .na
107 \\fB\\fB-D\\fR\\fR
108 .ad
109 .RS 15n
110 Formats a \\fB720KB\\fR (3.5 inch) or \\fB360KB\\fR (5.25 inch) double-density
111 diskette (same as the \\fB-l\\fR or \\fB-L\\fR options). This is the default for
112 double-density type drives. It is needed if the drive is a high- or
113 extended-density type.
114 .RE

116 .sp
117 .ne 2
118 .na
119 \\fB\\fB-e\\fR\\fR
120 .ad
121 .RS 15n
122 Ejects the diskette when done. This feature is not available on all systems.

```

```

123 .RE
125 .sp
126 .ne 2
127 .na
128 \fB\fB-E\fR\fR
129 .ad
130 .RS 15n
131 Formats a \fB2.88MB\fR (3.5 inch) extended-density diskette. This is the
132 default for extended-density type drives.
133 .RE
135 .sp
136 .ne 2
137 .na
138 \fB\fB-f\fR\fR
139 .ad
140 .RS 15n
141 Forces formatting, that is, this option does not ask for confirmation before
142 starting format.
143 .RE
145 .sp
146 .ne 2
147 .na
148 \fB\fB-H\fR\fR
149 .ad
150 .RS 15n
151 Formats a \fB1.44MB\fR (3.5 inch) or \fB1.2MB\fR (5.25 inch) high-density
152 diskette. This is the default for high-density type drives; it is needed if the
153 drive is the extended-density type.
154 .RE
156 .sp
157 .ne 2
158 .na
159 \fB\fB-M\fR\fR
160 .ad
161 .RS 15n
162 Writes a \fB1.2MB\fR (3.5 inch) medium-density format on a high-density
163 diskette (use only with the -t nec option). This is the same as using \fB-m\fR.
164 .sp
165 This feature is not available on all systems.
166 .RE
168 .sp
169 .ne 2
170 .na
171 \fB\fB-q\fR\fR
172 .ad
173 .RS 15n
174 Quiet; does not print status messages.
175 .RE
177 .sp
178 .ne 2
179 .na
180 \fB\fB-t\fR \fBdos\fR\fR
181 .ad
182 .RS 15n
183 Installs an \fBMS-DOS\fR file system and boot sector formatting. This is
184 equivalent to the \fBDOS\fR format command or the \fB-d\fR option.
185 .RE
187 .sp
188 .ne 2

```

```

189 .na
190 \fB\fB-t\fR \fBnec\fR\fR
191 .ad
192 .RS 15n
193 Installs an \fBNEC-DOS\fR file system and boot sector on the disk after
194 formatting. This should be used only with the \fB-M\fR option. This feature is
195 not available on all systems.
196 .RE
198 .sp
199 .ne 2
200 .na
201 \fB\fB-U\fR\fR
202 .ad
203 .RS 15n
204 Performs \fBumount\fR on any file systems and then formats. See
205 \fBmount\fR(1M).
206 .RE
208 .sp
209 .ne 2
210 .na
211 \fB\fB-v\fR\fR
212 .ad
213 .RS 15n
214 Verifies each block of the diskette after the format.
215 .RE
217 .sp
218 .ne 2
219 .na
220 \fB\fB-x\fR\fR
221 .ad
222 .RS 15n
223 Skips the format and only writes a SunOS label or an \fBMS-DOS\fR file system.
224 .RE
226 .SH OPERANDS
227 .sp
228 .LP
229 The following operands are supported:
230 .sp
231 .ne 2
232 .na
233 \fB\fIdevname\fR\fR
234 .ad
235 .RS 11n
236 Replaces \fIdevname\fR with \fBrdiskette0\fR (systems without volume
237 management) or \fBfloppy0\fR (systems with volume management) to use the first
238 drive or \fBrdiskette1\fR (systems without volume management) or \fBfloppy1\fR
239 (systems with volume management) to use the second drive. If \fIdevname\fR is
240 omitted, the first drive, if one exists, is used. For \fBPCMCIA\fR memory
241 cards, replace \fIdevname\fR with the device name for the \fBPCMCIA\fR memory
242 card which resides in \fB/dev/rdisk/c\fIIN\fRt\fIIN\fRd\fIIN\fRs\fIIN\fR or
243 \fB/dev/dsk/c\fIIN\fRt\fIIN\fRd\fIIN\fRs\fIIN\fR. If \fIdevname\fR is omitted, the
244 default diskette drive, if one exists, is used.
245 .sp
246 If devname is omitted, the default diskette drive, if one exists, will be used.
247 \fIIN\fR represents a decimal number and can be specified as follows:
248 .sp
249 .ne 2
250 .na
251 \fBc\fIIN\fR\fR
252 .ad
253 .RS 6n
254 Controller \fIIN\fR

```



```

255 .RE
257 .sp
258 .ne 2
259 .na
260 \fBt\fIN\fR\fR
261 .ad
262 .RS 6n
263 Technology type \fIN\fR:
264 .sp
265 .in +2
266 .nf

268 0x1      ROM
269 0x2      OTPROM
270 0x3      EPROM
271 0x4      EEPROM
272 0x5      FLASH
273 0x6      SRAM
274 0x7      DRAM
275 .fi
276 .in -2
277 .sp

279 .RE

281 .sp
282 .ne 2
283 .na
284 \fBd\fIN\fR\fR
285 .ad
286 .RS 6n
287 Technology region in type \fIN\fR.
288 .RE

290 .sp
291 .ne 2
292 .na
293 \fBs\fIN\fR\fR
294 .ad
295 .RS 6n
296 Slice \fIN\fR.
297 .RE

299 The following options are provided for compatibility with previous versions of
300 \fBfdformat\fR. Their use is discouraged.
301 .sp
302 .ne 2
303 .na
304 \fBd\fB-d\fR\fR
305 .ad
306 .RS 6n
307 Formats an \fBMS-DOS\fR floppy diskette or \fBPCMCIA\fR memory card (same as
308 \fB-t\fR \fBdos\fR). This is equivalent to the \fBMS-DOS FORMAT\fR command.
309 .RE

311 .sp
312 .ne 2
313 .na
314 \fBd\fB-l\fR\fR
315 .ad
316 .RS 6n
317 Formats a \fB720KB\fR (3.5 inch) or \fB360KB\fR (5.25 inch) double-density
318 diskette (same as \fB-D\fR or \fB-L\fR). This is the default for double-density
319 type drives; it is needed if the drive is the high- or extended-density type.
320 .RE

```

```

322 .sp
323 .ne 2
324 .na
325 \fBd\fB-L\fR\fR
326 .ad
327 .RS 6n
328 Formats a \fB720KB\fR (3.5 inch) or \fB360KB\fR (5.25 inch) double-density
329 diskette (same as \fB-l\fR or \fB-D\fR). This is the default for double-density
330 type drives.
331 .RE

333 .sp
334 .ne 2
335 .na
336 \fBd\fB-m\fR\fR
337 .ad
338 .RS 6n
339 Writes a \fB1.2 MB\fR (3.5 inch) medium- density format on a high-density
340 diskette (use only with the \fB-t\fR \fBnec\fR option). This is the same as
341 using \fB-M\fR. This feature is not available on all systems.
342 .RE

344 .RE

346 .SH FILES
347 .sp
348 .ne 2
349 .na
350 \fB/dev/diskette0\fR
351 .ad
352 .RS 24n
353 Directory providing block device access for the media in floppy drive \fB0\fR.
354 .RE

356 .sp
357 .ne 2
358 .na
359 \fB/dev/diskette0\fR
360 .ad
361 .RS 24n
362 Directory providing character device access for the media in floppy drive
363 \fB0\fR.
364 .RE

366 .sp
367 .ne 2
368 .na
369 \fB/dev/aliases/floppy0\fR
370 .ad
371 .RS 24n
372 Symbolic link to the character device for the media in floppy drive \fB0\fR.
373 .RE

375 .sp
376 .ne 2
377 .na
378 \fB/dev/rdiskette\fR
379 .ad
380 .RS 24n
381 Directory providing character device access for the media in the primary floppy
382 drive, usually drive \fB0\fR.
383 .RE

385 .sp
386 .ne 2

```

```

387 .na
388 \fB/dev/dsk/c\fIN\fRt\fIN\fRd\fIN\fRs\fIN\fR\
389 .ad
390 .RS 24n
391 Directory providing block device access for the \fBPCMCIA\fR memory card. See
392 OPERANDS for a description of \fIN\fR.
393 .RE

395 .sp
396 .ne 2
397 .na
398 \fB/dev/rdisk/c\fIN\fRt\fIN\fRd\fIN\fRs\fIN\fR\
399 .ad
400 .RS 24n
401 Directory providing character device access for the \fBPCMCIA\fR memory card.
402 See OPERANDS for a description of \fIN\fR.
403 .RE

405 .sp
406 .ne 2
407 .na
408 \fB/dev/aliases/pcmem\fIS\fR\
409 .ad
410 .RS 24n
411 Symbolic link to the character device for the \fBPCMCIA\fR memory card in
412 socket \fIS\fR where \fIS\fR represents a \fBPCMCIA\fR socket number.
413 .RE

415 .sp
416 .ne 2
417 .na
418 \fB/dev/rdisk/c\fIN\fRt\fIN\fRd\fIN\fRs\fIN\fR\
419 .ad
420 .RS 24n
421 Directory providing character device access for the \fBPCMCIA\fR memory card.
422 See OPERANDS for a description of \fIN\fR.
423 .RE

425 .sp
426 .ne 2
427 .na
428 \fB/dev/dsk/c\fIN\fRt\fIN\fRd\fIN\fRs\fIN\fR\
429 .ad
430 .RS 24n
431 Directory providing block device access for the \fBPCMCIA\fR memory card. See
432 OPERANDS for a description of \fIN\fR.
433 .RE

435 .SH SEE ALSO
436 .sp
437 .LP
438 \fBbcpio\fR(1), \fBbeject\fR(1), \fBbrmformat\fR(1), \fBbtar\fR(1),
439 \fBbvolcheck\fR(1), \fBbmount\fR(1M), \fBbnewfs\fR(1M), \fBbprtvtoc\fR(1M),
440 \fBbattributes\fR(5), \fBbpcfs\fR(7FS)
434 \fBbvolcheck\fR(1), \fBbvolrnmount\fR(1), \fBbmount\fR(1M), \fBbnewfs\fR(1M),
435 \fBbprtvtoc\fR(1M), \fBbattributes\fR(5), \fBbpcfs\fR(7FS)
441 .SS "x86 Only"
442 .sp
443 .LP
444 \fBbfd\fR(7D)
445 .SH NOTES
446 .sp
447 .LP
448 A diskette or \fBPCMCIA\fR memory card containing a \fBuffs\fR file system
449 created on a SPARC based system (by using \fBbfdformat\fR and \fBnewfs\fR(1M)),
450 is not identical to a diskette or \fBPCMCIA\fR memory card containing a ufs

```

```

451 file system created on an x86 based system. Do not interchange ufs diskettes or
452 memory cards between these platforms. Use \fBcpio\fR(1) or \fBtar\fR(1) to
453 transfer files on diskettes or memory cards between them. A diskette or
454 \fBPCMCIA\fR memory card formatted using the \fB-t\fR \fBdos\fR option (or
455 \fB-d\fR) for \fBMS-DOS\fR does not have the necessary system files, and is
456 therefore not bootable. Trying to boot from it on a \fBPC\fR produces the
457 following message:
458 .sp
459 .in +2
460 .nf
461 Non-System disk or disk error.
462 Replace and strike any key when ready
463 .fi
464 .in -2
465 .sp

467 .SH BUGS
468 .sp
469 .LP
470 Currently, bad sector mapping is not supported on floppy diskettes or
471 \fBPCMCIA\fR memory cards. Therefore, a diskette or memory card is unusable if
472 \fBfdformat\fR finds an error (bad sector).

```

new/usr/src/man/man1/rmformat.1

1

```
*****
14917 Sun May 4 18:28:38 2014
new/usr/src/man/man1/rmformat.1
4833 Remove volrmount
4845 rm(u)mount don't always print mount/unmount errors
4846 HAL partition names don't match real partition names
Reviewed by: Dan McDonald <danmcd@omniti.com>
Reviewed by: Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
*****
1 \" te
2 .\" Copyright (c) 2009, Sun Microsystems, Inc. All Rights Reserved.
3 .\" Copyright 2014 Andrew Stormont.
4 #endif /* ! codereview */
5 .\" The contents of this file are subject to the terms of the Common Development
6 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
7 .\" When distributing Covered Code, include this CDDL HEADER in each file and in
8 .TH RMFORMAT 1 \"May 2, 2014\"
9 .TH RMFORMAT 1 \"Feb 19, 2009\"
10 rmformat \- removable rewritable media format utility
11 .SH SYNOPSIS
12 .LP
13 .nf
14 \fBrmformat\fR [\fB-DeHuv\fR] [\fB-b\fR \fIlabel\fR] [\fB-c\fR \fIblockno\fR]
15 [\fB-F\fRquick | long | force ] [\fB-s\fR \fIfilename\fR] [\fIdevname\fR]
16 .fi
18 .LP
19 .nf
20 \fBrmformat\fR \fB-V\fR read | write \fIdevname\fR
21 .fi
23 .LP
24 .nf
25 \fBrmformat\fR \fB-l\fR [\fIdevname\fR]
26 .fi
28 .SH DESCRIPTION
29 .sp
30 .LP
31 The \fBrmformat\fR utility is used to format, label, partition, and perform
32 other miscellaneous functions on removable, rewritable media that include
33 floppy drives, and the \fBPCMCIA\fR memory and \fBata\fR cards. The
34 \fBrmformat\fR utility should also be used with all USB mass storage devices,
35 including USB hard drives. This utility can also be used for the verification
36 and surface analysis and for repair of the bad sectors found during
37 verification if the drive or the driver supports bad block management.
38 .sp
39 .LP
40 After formatting, \fBrmformat\fR writes the label, which covers the full
41 capacity of the media as one slice on floppy and \fBPCMCIA\fR memory cards to
42 maintain compatibility with the behavior of \fBfdformat\fR. The partition
43 information can be changed with the help of other options provided by
44 \fBrmformat\fR.
45 .SH OPTIONS
46 .sp
47 .LP
48 The following options are supported:
49 .sp
50 .ne 2
51 .na
52 \fB-b\fR \fIlabel\fR
53 .ad
54 .sp .6
55 .RS 4n
56 Labels the media with a SUNOS label. A SUNOS volume label name is restricted to
```

new/usr/src/man/man1/rmformat.1

2

```
57 8 characters. For media size greater than 1 TB, an EFI label is created. For
58 writing a \fBDOS\fR Volume label, the user should use \fBmkfs_pcfs\fR(1M).
59 .RE
61 .sp
62 .ne 2
63 .na
64 \fB-b\fR \fIblockno\fR
65 .ad
66 .sp .6
67 .RS 4n
68 Corrects and repairs the given block. This correct and repair option may not be
69 applicable to all devices supported by \fBrmformat\fR, as some devices may have
70 a drive with bad block management capability and others may have this option
71 implemented in the driver. If the drive or driver supports bad block
72 management, a best effort is made to rectify the bad block. If the bad block
73 still cannot be rectified, a message is displayed to indicate the failure to
74 repair. The block number can be provided in decimal, octal, or hexadecimal
75 format.
76 .sp
77 The normal floppy and \fBPCMCIA\fR memory and ata cards do not support bad
78 block management.
79 .RE
81 .sp
82 .ne 2
83 .na
84 \fB-D\fR
85 .ad
86 .sp .6
87 .RS 4n
88 Formats a 720KB (3.5 inch) double density diskette. This is the default for
89 double density type drives. This option is needed if the drive is a high or
90 extended-density type.
91 .RE
93 .sp
94 .ne 2
95 .na
96 \fB-e\fR
97 .ad
98 .sp .6
99 .RS 4n
100 Ejects the media upon completion. This feature may not be available if the
101 drive does not support motorized eject.
102 .RE
104 .sp
105 .ne 2
106 .na
107 \fB-F\fR quick | long | force
108 .ad
109 .sp .6
110 .RS 4n
111 Formats the media.
112 .sp
113 The \fBquick\fR option starts a format without certification or format with
114 limited certification of certain tracks on the media.
115 .sp
116 The \fBlong\fR option starts a complete format. For some devices this might
117 include the certification of the whole media by the drive itself.
118 .sp
119 The \fBforce\fR option to format is provided to start a long format without
120 user confirmation before the format is started.
121 .sp
122 In legacy media such as floppy drives, all options start a long format
```

123 depending on the mode (Extended Density mode, High Density mode, or Double
124 Density mode) with which the floppy drive operates by default. On \fBPCMCIA\fR
125 memory cards, all options start a long format.
126 .RE

128 .sp
129 .ne 2
130 .na

131 \fB\fB-H\fR\fR

132 .ad
133 .sp .6
134 .RS 4n
135 Formats a 1.44 MB (3.5 inch) high density diskette. This is the default for
136 high density type drives. It is needed if the drive is the Extended Density
137 type.
138 .RE

140 .sp
141 .ne 2
142 .na

143 \fB\fB-l\fR\fR

144 .ad
145 .sp .6
146 .RS 4n
147 Lists all removable devices. By default, without any options, \fBrmformat\fR
148 also lists all removable devices. If the \fBdev_name\fR is given,
149 \fBrmformat\fR lists the device associated with the \fBdev_name\fR. The output
150 shows the device pathname, vendor information, and the device type.
151 .RE

153 .sp
154 .ne 2
155 .na

156 \fB\fB-s\fR \fIfilename\fR\fR

157 .ad
158 .sp .6
159 .RS 4n
160 Enables the user to lay out the partition information in the SUNOS label.
161 .sp

162 The user should provide a file as input with information about each slice in a
163 format providing byte offset, size required, tags, and flags, as follows:

164 .sp
165 .in +2
166 .nf
167 slices: \fIn\fR = \fIoffset\fR, \fIsize\fR [, \fIflags\fR, \fItags\fR]
168 .fi
169 .in -2
170 .sp

172 where \fIn\fR is the slice number, \fIoffset\fR is the byte offset at which the
173 slice \fIn\fR starts, and \fIsize\fR is the required size for slice \fIn\fR.
174 Both \fIoffset\fR and \fIsize\fR must be a multiple of 512 bytes. These numbers
175 can be represented as decimal, hexadecimal, or octal numbers. No floating point
176 numbers are accepted. Details about maximum number of slices can be obtained
177 from the \fISystem Administration Guide: Basic Administration\fR.

178 .sp
179 To specify the \fIsize\fR or \fIoffset\fR in kilobytes, megabytes, or
180 gigabytes, add \fBKB\fR, \fBMB\fR, \fBGB\fR, respectively. A number without a
181 suffix is assumed to be a byte offset. The flags are represented as follows:

182 .sp
183 .in +2
184 .nf
185 \fBwm\fR = read-write, mountable
186 \fBwu\fR = read-write, unmountable
187 \fBru\fR = read-only, unmountable
188 .fi

189 .in -2
190 .sp

192 The tags are represented as follows: \fBunassigned\fR, \fBboot\fR, \fBroot\fR,
193 \fBswap\fR, \fBusr\fR, \fBbackup\fR, \fBstand\fR, \fBvar\fR, \fBhome\fR,
194 \fBalternates\fR.

195 .sp
196 The tags and flags can be omitted from the four tuple when finer control on
197 those values is not required. It is required to omit both or include both. If
198 the tags and flags are omitted from the four tuple for a particular slice, a
199 default value for each is assumed. The default value for flags is \fBwm\fR and
200 for tags is \fBunassigned\fR.

201 .sp
202 Either full tag names can be provided or an abbreviation for the tags can be
203 used. The abbreviations can be the first two or more letters from the standard
204 tag names. \fBrmformat\fR is case insensitive in handling the defined tags &
205 flags.

206 .sp
207 Slice specifications are separated by :

208 .sp
209 For example:

210 .sp
211 .in +2
212 .nf
213 slices: 0 = 0, 30MB, "wm", "home" :
214 1 = 30MB, 51MB :
215 2 = 0, 100MB, "wm", "backup" :
216 6 = 81MB, 19MB

217 .fi
218 .in -2
219 .sp

221 \fBrmformat\fR does the necessary checking to detect any overlapping partitions
222 or illegal requests to addresses beyond the capacity of the media under
223 consideration. There can be only one slice information entry for each slice
224 \fIn\fR. If multiple slice information entries for the same slice \fIn\fR are
225 provided, an appropriate error message is displayed. The slice \fB2\fR is the
226 backup slice covering the whole disk capacity. The pound sign character,
227 \fB#\fR, can be used to describe a line of comments in the input file. If the
228 line starts with \fB#\fR, then \fBrmformat\fR ignores all the characters
229 following \fB#\fR until the end of the line.

230 .sp
231 Partitioning some of the media with very small capacity is permitted, but be
232 cautious in using this option on such devices.
233 .RE

235 .sp
236 .ne 2
237 .na

238 \fB\fB-U\fR\fR

239 .ad
240 .sp .6
241 .RS 4n
242 Performs \fBumount\fR on any file systems and then formats. See
243 \fBmount\fR(1M). This option unmounts all the mounted slices and issues a long
244 format on the device requested.
245 .RE

247 .sp
248 .ne 2
249 .na

250 \fB\fB-V\fR read | write\fR

251 .ad
252 .sp .6
253 .RS 4n
254 Verifies each block of media after format. The write verification is a

```

255 destructive mechanism. The user is queried for confirmation before the
256 verification is started. The output of this option is a list of block numbers,
257 which are identified as bad.
258 .sp
259 The read verification only verifies the blocks and report the blocks which are
260 prone to errors.
261 .sp
262 The list of block numbers displayed can be used with the \fB-c\fR option for
263 repairing.
264 .RE

266 .SH OPERANDS
267 .sp
268 .LP
269 The following operand is supported:
270 .sp
271 .ne 2
272 .na
273 \fB\fIdevname\fR\fR
274 .ad
275 .sp .6
276 .RS 4n
277 \fIdevname\fR can be provided as absolute device pathname or relative pathname
278 for the device from the current working directory or the nickname, such as
279 \fBcdrom\fR or \fBrdisk\fR.
280 .sp
281 For floppy devices, to access the first drive use \fB/dev/rdiskette0\fR (for
282 systems without volume management) or \fBfloppy0\fR (for systems with volume
283 management). Specify \fB/dev/rdiskette1\fR (for systems without volume
284 management) or \fBfloppy1\fR (for systems with volume management) to use the
285 second drive.
286 .sp
287 For systems without volume management running, the user can also provide the
288 absolute device pathname as \fB/dev/rdisk/c\fI?fRt\fI?fRd\fI?fRs\fI?fR\fR or
289 the appropriate relative device pathname from the current working directory.
290 .RE

292 .SH EXAMPLES
293 .LP
294 \fBExample 1 \fRFormatting a Diskette
295 .sp
296 .in +2
297 .nf
298 example$ \fBrmformat -F quick /dev/rdiskette\fR
299 Formatting will erase all the data on disk.
300 Do you want to continue? (y/n)\fBy\fR
301 .fi
302 .in -2
303 .sp

305 .LP
306 \fBExample 2 \fRFormatting a Diskette for a UFS File System
307 .sp
308 .LP
309 The following example formats a diskette and creates a UFS file system:

311 .sp
312 .in +2
313 .nf
314 example$ \fBrmformat -F quick /dev/aliases/floppy0\fR
315 Formatting will erase all the data on disk.
316 Do you want to continue? (y/n)\fBy\fR
317 example$ \fBsu\fR
318 # \fB/usr/sbin/newfs /dev/aliases/floppy0\fR
319 newfs: construct a new file system /dev/rdiskette: (y/n)? \fBy\fR
320 /dev/rdiskette: 2880 sectors in 80 cylinders of 2 tracks, 18 sectors

```

```

321          1.4MB in 5 cyl groups (16 c/g, 0.28MB/g, 128 i/g)
322 super-block backups (for fsck -F ufs -o b=#) at:
323 32, 640, 1184, 1792, 2336,
324 #
325 .fi
326 .in -2
327 .sp

329 .LP
330 \fBExample 3 \fRFormatting Removable Media for a PCFS File System
331 .sp
332 .LP
333 The following example shows how to create an alternate \fBfdisk\fR partition:

335 .sp
336 .in +2
337 .nf
338 example$ \fBrmformat -F quick /dev/rdisk/c0t4d0s2:c\fR
339 Formatting will erase all the data on disk.
340 Do you want to continue? (y/n)\fBy\fR
341 example$ \fBsu\fR
342 # \fBfdisk /dev/rdisk/c0t4d0s2:c\fR
343 # \fBmkfs -F pcfs /dev/rdisk/c0t4d0s2:c\fR
344 Construct a new FAT file system on /dev/rdisk/c0t4d0s2:c: (y/n)? \fBy\fR
345 #
346 .fi
347 .in -2
348 .sp

350 .sp
351 .LP
352 The following example describes how to create a \fBPCFS\fR file system
353 \fBwithout\fR an \fBfdisk\fR partition:

355 .sp
356 .in +2
357 .nf
358 example$ \fBrmformat -F quick /dev/rdiskette\fR
359 Formatting will erase all the data on disk.
360 Do you want to continue? (y/n)\fBy\fR
361 example$ \fBsu\fR
362 # \fBmkfs -F pcfs -o nofdisk,size=2 /dev/rdiskette\fR
363 Construct a new FAT file system on /dev/rdiskette: (y/n)? \fBy\fR
364 #
365 .fi
366 .in -2
367 .sp

369 .LP
370 \fBExample 4 \fRListing All Removable Devices
371 .sp
372 .LP
373 The following example shows how to list removable devices. This output shows a
374 long listing of such devices.

376 .sp
377 .in +2
378 .nf
379 example$ rmformat -l
380 Looking for devices...
381 Logical Node: /dev/rdisk/c5t0d0s2
382 Physical Node: /pci@1e,600000/usb@b/hub@2/storage@4/disk@0,0
383 Connected Device: TEAC      FD-05PUB      1026
384 Device Type: Floppy drive
385 Bus: USB
386 Size: 1.4 MB

```

```

387 Label: floppy
388 Access permissions: Medium is not write protected.
389 .fi
390 .in -2
391 .sp

393 .SH FILES
394 .sp
395 .ne 2
396 .na
397 \fB\fB/dev/diskette0\fR\fR
398 .ad
399 .sp .6
400 .RS 4n
401 Directory providing block device access for the media in floppy drive 0.
402 .RE

404 .sp
405 .ne 2
406 .na
407 \fB\fB/dev/rdiskette0\fR\fR
408 .ad
409 .sp .6
410 .RS 4n
411 Directory providing character device access for the media in floppy drive 0.
412 .RE

414 .sp
415 .ne 2
416 .na
417 \fB\fB/dev/aliases\fR\fR
418 .ad
419 .sp .6
420 .RS 4n
421 Directory providing symbolic links to the character devices for the different
422 media under the control of volume management using appropriate alias.
423 .RE

425 .sp
426 .ne 2
427 .na
428 \fB\fB/dev/aliases/floppy0\fR\fR
429 .ad
430 .sp .6
431 .RS 4n
432 Symbolic link to the character device for the media in floppy drive 0.
433 .RE

435 .sp
436 .ne 2
437 .na
438 \fB\fB/dev/rdiskette\fR\fR
439 .ad
440 .sp .6
441 .RS 4n
442 Symbolic link providing character device access for the media in the primary
443 floppy drive, usually drive 0.
444 .RE

446 .sp
447 .ne 2
448 .na
449 \fB\fB/dev/dsk\fR\fR
450 .ad
451 .sp .6
452 .RS 4n

```

```

453 Directory providing block device access for the \fBPCMCIA\fR memory and ata
454 cards and removable media devices.
455 .RE

457 .sp
458 .ne 2
459 .na
460 \fB\fB/dev/rdsk\fR\fR
461 .ad
462 .sp .6
463 .RS 4n
464 Directory providing character device access for the \fBPCMCIA\fR memory and ata
465 cards and removable media devices.
466 .RE

468 .sp
469 .ne 2
470 .na
471 \fB\fB/dev/aliases/pcmemS\fR\fR
472 .ad
473 .sp .6
474 .RS 4n
475 Symbolic link to the character device for the \fBPCMCIA\fR memory card in
476 socket S, where S represents a \fBPCMCIA\fR socket number.
477 .RE

479 .sp
480 .ne 2
481 .na
482 \fB\fB/dev/aliases/rmdisk0\fR\fR
483 .ad
484 .sp .6
485 .RS 4n
486 Symbolic link to the generic removable media device that is not a \fBCD-ROM\fR,
487 floppy, \fBDVD-ROM\fR, \fBPCMCIA\fR memory card, and so forth.
488 .RE

490 .sp
491 .ne 2
492 .na
493 \fB\fB/dev/rdsk\fR\fR
494 .ad
495 .sp .6
496 .RS 4n
497 Directory providing character device access for the \fBPCMCIA\fR memory and ata
498 cards and other removable devices.
499 .RE

501 .sp
502 .ne 2
503 .na
504 \fB\fB/dev/dsk\fR\fR
505 .ad
506 .sp .6
507 .RS 4n
508 Directory providing block device access for the \fBPCMCIA\fR memory and ata
509 cards and other removable media devices.
510 .RE

512 .SH SEE ALSO
513 .sp
514 .LP
515 \fB\bcpio\fR(1), \fB\bect\fR(1), \fB\bfdformat\fR(1), \fB\bstar\fR(1),
516 \fB\bvolcheck\fR(1), \fB\bformat\fR(1M), \fB\bmkfs_pcfs\fR(1M), \fB\bmount\fR(1M),
517 \fB\bnewfs\fR(1M), \fB\bprtvtoc\fR(1M), \fB\bmount\fR(1M), \fB\brcp.smserved\fR(1M),
518 \fB\battributes\fR(5), \fB\bcsa2usb\fR(7D), \fB\bfsd\fR(7D), \fB\bpcfs\fR(7FS),

```

```
519 \fBuds\fr(7FS)
511 \fBvolcheck\fr(1), \fBvolrmount\fr(1), \fBformat\fr(1M), \fBmkfs_pcfs\fr(1M),
512 \fBmount\fr(1M), \fBnewfs\fr(1M), \fBprtvtoc\fr(1M), \fBrmount\fr(1M),
513 \fBrpc.smserverd\fr(1M), \fBattributes\fr(5), \fBscsa2usb\fr(7D), \fBsd\fr(7D),
514 \fBpcfs\fr(7FS), \fBuds\fr(7FS)
520 .sp
521 .LP
522 \fISystem Administration Guide: Basic Administration\fr
523 .SH NOTES
524 .sp
525 .LP
526 A rewritable media or \fBPCMCIA\fr memory card or \fBPCMCIA\fr ata card
527 containing a \fBuds\fr file system created on a SPARC-based system (using
528 \fBnewfs\fr(1M)) is not identical to a rewritable media or \fBPCMCIA\fr memory
529 card containing a \fBuds\fr file system created on an x86 based system. Do not
530 interchange any removable media containing \fBuds\fr between these platforms;
531 use \fBcpio\fr(1) or \fBtar\fr(1) to transfer files on diskettes or memory
532 cards between them. For interchangeable filesystems refer to \fBpcfs\fr(7FS)
533 and \fBuds\fr(7FS).
534 .sp
535 .LP
536 \fBrmformat\fr might not list all removable devices in virtualization
537 environments.
538 .SH BUGS
539 .sp
540 .LP
541 Currently, bad sector mapping is not supported on floppy diskettes or
542 \fBPCMCIA\fr memory cards. Therefore, a diskette or memory card is unusable if
543 \fBrmformat\fr finds an error (\fBbad sector\fr).
```

new/usr/src/man/man1m/rmmount.1m

1

5890 Sun May 4 18:28:38 2014

new/usr/src/man/man1m/rmmount.1m

4833 Remove volrmmount

4845 rm(u)mount don't always print mount/unmount errors

4846 HAL partition names don't match real partition names

Reviewed by: Dan McDonald <danmcd@omniti.com>

Reviewed by: Josef 'Jeff' Sipek <jeffpc@josefsipek.net>

```
1 \" te
2.\" Copyright (c) 2004, Sun Microsystems, Inc. All Rights Reserved
3.\" Copyright 2014 Andrew Stormont.
4 #endif /* ! codereview */
5.\" The contents of this file are subject to the terms of the Common Development
6.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
7.\" When distributing Covered Code, include this CDDL HEADER in each file and in
8.TH RMMOUNT 1M \"May 2, 2014\"
9.TH RMMOUNT 1M \"Mar 1, 2007\"
10 .SH NAME
11 rmmount \- removable media mounter for CD-ROM, floppy, Jaz drive, and others
12 .SH SYNOPSIS
13 .LP
14 \fB/usr/sbin/rmmount\fR [\fB-D\fR]
15 .fi
17 .SH DESCRIPTION
18 .sp
19 .LP
20 The \fBBrmmount\fR utility is a removable media mounter that is executed by
21 volume management whenever a removable medium, such as a \fBBCD-ROM\fR or a
22 floppy, is inserted. Removable media is managed by an application or a volume
23 manager.
24 manager. \fBBrmmount\fR can also be called by using \fBVolrmmount\fR(1).
25 .sp
26 Upon insertion of a medium and following invocation of the \fBVolcheck\fR(1)
27 command, \fBBrmmount\fR determines what type of file system (if any) is on that
28 medium. If a file system is present, \fBBrmmount\fR mounts the file system in
29 one of the locations listed below.
30 .sp
31 .LP
32 For a diskette (floppy):
33 .sp
34 .ne 2
35 .na
36 \fB\fB\fB/floppy/floppy0\fR\fR
37 .ad
38 .RS 26n
39 symbolic link to mounted floppy in local floppy drive
40 .RE
42 .sp
43 .ne 2
44 .na
45 \fB\fB\fB/floppy/floppy_name\fR\fR
46 .ad
47 .RS 26n
48 mounted named floppy
49 .RE
51 .sp
52 .ne 2
53 .na
54 \fB\fB\fB/floppy/unnamed_floppy\fR\fR
55 .ad
```

new/usr/src/man/man1m/rmmount.1m

2

```
56 .RS 26n
57 mounted unnamed floppy
58 .RE
60 .sp
61 .LP
62 For a CD-ROM or a DVD-ROM:
63 .sp
64 .ne 2
65 .na
66 \fB\fB\fB/cdrom/cdrom0\fR\fR
67 .ad
68 .sp .6
69 .RS 4n
70 symbolic link to mounted \fBBCD-ROM\fR in local \fBBCD-ROM\fR drive
71 .RE
73 .sp
74 .ne 2
75 .na
76 \fB\fB\fB/cdrom/CD-ROM_name\fR\fR
77 .ad
78 .sp .6
79 .RS 4n
80 mounted named \fBBCD-ROM\fR
81 .RE
83 .sp
84 .ne 2
85 .na
86 \fB\fB\fB/cdrom/CD-ROM_name/partition\fR\fR
87 .ad
88 .sp .6
89 .RS 4n
90 mounted named \fBBCD-ROM\fR with partitioned file system
91 .RE
93 .sp
94 .ne 2
95 .na
96 \fB\fB\fB/cdrom/unnamed_cdrom\fR\fR
97 .ad
98 .sp .6
99 .RS 4n
100 mounted unnamed \fBBCD-ROM\fR
101 .RE
103 .sp
104 .LP
105 For a Zip drive:
106 .sp
107 .ne 2
108 .na
109 \fB\fB\fB/rmdisk/zip0\fR\fR
110 .ad
111 .RS 30n
112 symbolic link to mounted Zip medium in local Zip drive
113 .RE
115 .sp
116 .ne 2
117 .na
118 \fB\fB\fB/rmdisk/\fIFZip_name\fR\fR\fR
119 .ad
120 .RS 30n
121 mounted named Zip medium
```



```

122 .RE

124 .sp
125 .ne 2
126 .na
127 \fB\fB/rmdisk/\fIZip_name\fR/partition\fR\fR
128 .ad
129 .RS 30n
130 mounted named Zip medium with partitioned file system
131 .RE

133 .sp
134 .ne 2
135 .na
136 \fB\fB/rmdisk/unnamed_zip\fR\fR
137 .ad
138 .RS 30n
139 mounted unnamed Zip medium
140 .RE

142 .sp
143 .LP
144 For a Jaz drive:
145 .sp
146 .ne 2
147 .na
148 \fB\fB/rmdisk/jaz0\fR\fR
149 .ad
150 .RS 30n
151 symbolic link to mounted Jaz medium in local Jaz drive
152 .RE

154 .sp
155 .ne 2
156 .na
157 \fB\fB/rmdisk/\fIJaz_name\fR\fR\fR
158 .ad
159 .RS 30n
160 mounted named Jaz medium
161 .RE

163 .sp
164 .ne 2
165 .na
166 \fB\fB/rmdisk/\fIJaz_name\fR/partition\fR\fR
167 .ad
168 .RS 30n
169 mounted named Jaz medium with partitioned file system
170 .RE

172 .sp
173 .ne 2
174 .na
175 \fB\fB/rmdisk/unnamed_Jaz\fR\fR
176 .ad
177 .RS 30n
178 mounted unnamed Jaz medium
179 .RE

181 .sp
182 .LP
183 For a generic "rmdisk" drive:
184 .sp
185 .ne 2
186 .na
187 \fB\fB/rmdisk/rmdisk0\fR\fR

```

```

188 .ad
189 .sp .6
190 .RS 4n
191 symbolic link to mounted removable medium in local removable medium drive
192 .RE

194 .sp
195 .ne 2
196 .na
197 \fB\fB/rmdisk/\fIrmdisk_name\fR\fR\fR
198 .ad
199 .sp .6
200 .RS 4n
201 mounted named removable medium
202 .RE

204 .sp
205 .ne 2
206 .na
207 \fB\fB/rmdisk/\fIrmdisk_name\fR/partition\fR\fR
208 .ad
209 .sp .6
210 .RS 4n
211 mounted named removable medium with partitioned file system
212 .RE

214 .sp
215 .ne 2
216 .na
217 \fB\fB/rmdisk/unnamed_rmdisk\fR\fR
218 .ad
219 .sp .6
220 .RS 4n
221 mounted unnamed removable medium
222 .RE

224 .sp
225 .LP
226 If the media is read-only (for example, a \fBCD-ROM\fR or a floppy with
227 write-protect tab set), the file system is mounted read-only.
228 .sp
229 .LP
230 If a file system is not identified, \fBBrmmount\fR does not mount a file system.
231 See the \fI\fR for more information on the location of \fBCD-ROM\fR, floppy,
232 and other media without file systems.
233 .sp
234 .LP
235 If a file system type has been determined, it is then checked to see that it is
236 "clean." If the file system is "dirty," \fBfscck\fR \fB-p\fR (see
237 \fBfscck\fR(1M)) is run in an attempt to clean it. If \fBfscck\fR fails, the file
238 system is mounted read-only.
239 .sp
240 .LP
241 After the mount is complete, "actions" associated with the media type are
242 executed. These actions allow for the notification to other programs that new
243 media are available.
244 .sp
245 .LP
246 Actions are executed in the order in which they appear in the configuration
247 file. The action function can return either \fB1\fR or \fB0\fR. If it returns
248 \fB0\fR, no further actions will be executed. This allows the function to
249 control which applications are executed.
250 .sp
251 .LP
252 In order to execute an action, \fBBrmmount\fR performs a \fBdlopen\fR(3C) on the
253 shared object and calls the action function defined within it. The definition

```

```
254 of the interface to actions can be found in \fB/usr/include/rmmount.h\fR.
255 .sp
256 .LP
257 File systems mounted by \fBBrmmount\fR are always mounted with the \fBnosuid\fR
258 flag set, thereby disabling setuid programs and access to block or character
259 devices in that file system. Upon ejection, \fBBrmmount\fR unmounts mounted file
260 systems and executes actions associated with the media type. If a file system
261 is "busy" (that is, it contains the current working directory of a live
262 process), the ejection will fail.
263 .SH OPTIONS
264 .sp
265 .ne 2
266 .na
267 \fB\fB-D\fR\fR
268 .ad
269 .RS 6n
270 Turn on the debugging output from the \fBBrmmount\fR \fBdprintf\fR calls.
271 .RE

273 .SH FILES
274 .sp
275 .ne 2
276 .na
277 \fB/usr/lib/rmmount/*.so.1\fR
278 .ad
279 .RS 27n
280 shared objects used by \fBBrmmount\fR.
281 .RE

283 .SH SEE ALSO
284 .sp
285 .LP
286 \fBbvolcheck\fR(1), \fBfsck\fR(1M), \fBbdlopen\fR(3C),
287 \fBbvolcheck\fR(1), \fBvolrmmount\fR(1), \fBfsck\fR(1M), \fBbdlopen\fR(3C),
288 \fBattributes\fR(5)
288 .sp
289 .LP
290 \fI\fR
```

```

*****
2802 Sun May 4 18:28:38 2014
new/usr/src/pkg/manifests/service-storage-media-volume-manager.mf
4833 Remove volrmmount
4845 rm(u)mount don't always print mount/unmount errors
4846 HAL partition names don't match real partition names
Reviewed by: Dan McDonald <danmcd@omniti.com>
Reviewed by: Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
*****
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 #
22 #
23 # Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
24 # Copyright 2012 Nexenta Systems, Inc. All rights reserved.
25 # Copyright 2014 Andrew Stormont.
26 #endif /* ! codereview */
27 #
28 #
29 set name=pkg.fmri value=pkg:/service/storage/media-volume-manager@$(PKGVERS)
30 set name=pkg.description value="Non-graphical removable volume manager"
31 set name=pkg.summary value="Removable volume manager"
32 set name=info.classification value=org.opensolaris.category.2008:System/Media
33 set name=variant.arch value=$(ARCH)
34 dir path=lib variant.opensolaris.zone=global
35 dir path=lib/svc variant.opensolaris.zone=global
36 dir path=lib/svc/manifest group=sys variant.opensolaris.zone=global
37 dir path=lib/svc/manifest/system group=sys variant.opensolaris.zone=global
38 dir path=lib/svc/manifest/system/filesystem group=sys \
39   variant.opensolaris.zone=global
40 dir path=lib/svc/method variant.opensolaris.zone=global
41 dir path=usr group=sys
42 dir path=usr/bin
43 dir path=usr/lib
44 dir path=usr/sbin
45 dir path=usr/share/man/man1
46 dir path=usr/share/man/man1m
47 file path=lib/svc/manifest/system/filesystem/rmvolmgr.xml group=sys mode=0444 \
48   variant.opensolaris.zone=global
49 file path=lib/svc/method/svc-rmvolmgr mode=0555 \
50   variant.opensolaris.zone=global
51 file path=usr/bin/rmformat mode=4555
52 file path=usr/bin/rmmount mode=0555
53 file path=usr/bin/volcheck mode=0555
54 file path=usr/bin/volrmmount mode=0555
55 file path=usr/lib/rmvolmgr mode=0555
56 file path=usr/share/man/man1/rmformat.1
57 file path=usr/share/man/man1/rmmount.1

```

```

57 file path=usr/share/man/man1/volcheck.1
58 file path=usr/share/man/man1m/rmmount.1m
59 file path=usr/share/man/man1m/rmvolmgr.1m
60 legacy pkg=SUNWrmvolmgr desc="Non-graphical removable volume manager" \
61   name="Removable volume manager"
62 legacy pkg=SUNWrmvolmgr desc="Non-graphical removable volume manager (Root)" \
63   name="Removable volume manager (Root)"
64 license cr_Sun license=cr_Sun
65 license lic_CDDL license=lic_CDDL
66 link path=usr/bin/rmmount target=./rmmount
67 link path=usr/sbin/rmmount target=./bin/rmmount
68 link path=usr/share/man/man1/rmmount.1 target=rmmount.1

```