****
904 Sun Oct 8 15:35:27 2017
new/usr/src/cmd/diskinfo/Makefile
8708 Want diskinfo(1m) to list all disk bays, including those don't have disk in
****

```
   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 at
   9 # http://www.illumos.org/license/CDDL.
  10 #

  12 #
  13 # Copyright 2016 Nexenta Systems, Inc.
  14 #

  16 PROG=           diskinfo
  17 OBJS=           diskinfo.o
  18 SRCS=           $(OBJS:%.o=%.c)

  20 include         $(SRC)/cmd/Makefile.cmd
  21 include         $(SRC)/cmd/Makefile.ctf

  23 C99MODE=        $(C99_ENABLE)

  25 CPPFLAGS +=     -I$(SRC)/lib/fm/topo

  27 LDLIBS +=       -L$(ROOT)/usr/lib/fm -R/usr/lib/fm -ldiskmgt -lnvpair -ltopo \
  28     -lcmdutils
  27 LDLIBS +=       -L$(ROOT)/usr/lib/fm -R/usr/lib/fm -ldiskmgt -lnvpair -ltopo

  30 .KEEP_STATE:

  32 all:            $(PROG)

  34 $(PROG):        $(OBJS)
  35                 $(LINK.c) $(OBJS) -o $@ $(LDLIBS)
  36                 $(POST_PROCESS)

  38 clean:
  39                 $(RM) $(OBJS)

  41 install:        all $(ROOTPROG)

  43 include         $(SRC)/cmd/Makefile.targ
```

```
**********************************************************
   13712 Sun Oct  8 15:35:27 2017
new/usr/src/cmd/diskinfo/diskinfo.c
8708 Want diskinfo(1m) to list all disk bays, including those don't have disk in
**********************************************************
```

```
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */

/*
 * Copyright (c) 2013 Joyent Inc., All rights reserved.
 */

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <limits.h>
#include <assert.h>
#include <ctype.h>
#include <stdarg.h>
#include <strings.h>

#include <libdiskmgt.h>
#include <sys/nvpair.h>
#include <sys/param.h>
#include <sys/ccompile.h>
#include <sys/list.h>

#include <fm/libtopo.h>
#include <fm/topo_hc.h>
#include <fm/topo_list.h>
#include <sys/fm/protocol.h>
#include <modules/common/disk/disk.h>

typedef struct di_opts {
        boolean_t di_allslots;
        boolean_t di_scripted;
        boolean_t di_parseable;
        boolean_t di_physical;
        boolean_t di_condensed;
} di_opts_t;

static list_t g_disks;
static di_opts_t g_opts = { B_FALSE, B_FALSE, B_FALSE, B_FALSE, B_FALSE };

typedef struct di_phys {
        uint64_t dp_size;
        uint32_t dp_blksize;
        char *dp_vid;
        char *dp_pid;
        boolean_t dp_removable;
        boolean_t dp_ssd;
        char *dp_dev;
        char *dp_ctype;
        char *dp_serial;
```



```
        char *dp_slotname;
        const char *dp_dev;
        const char *dp_serial;
        const char *dp_slotname;
        int dp_chassis;
        int dp_slot;
        int dp_faulty;
        int dp_locate;
        list_node_t dp_next;
} di_phys_t;
```

*unchanged_portion_omitted_*

```
static void *
safe_zmalloc(size_t size)
{
        void *ptr = malloc(size);
        if (ptr == NULL)
                fatal(-1, "failed to allocate memeory");
        memset(ptr, 0, size);
        return (ptr);
}

static char *
safe_strdup(const char *s1)
{
        char *s2 = strdup(s1);
        if (s2 == NULL)
                fatal(-1, "failed to allocate memeory");
        return (s2);
}

static int
safe_asprintf(char **ret, const char *fmt, ...)
{
        va_list ap;
        int v;

        va_start(ap, fmt);
        v = vasprintf(ret, fmt, ap);
        va_end(ap);
        if (v == -1)
                fatal(-1, "failed to allocate memeory");
        return (v);
}

static void
usage(const char *execname)
{
        (void) fprintf(stderr, "Usage: %s [-aHp] [{-c|-P}]\n", execname);
        (void) fprintf(stderr, "Usage: %s [-Hp] [{-c|-P}]\n", execname);
}
```

*unchanged_portion_omitted_*

```
static void
set_disk_bay_info(topo_hdl_t *hp, tnode_t *np, di_phys_t *dip)
static int
disk_walker(topo_hdl_t *hp, tnode_t *np, void *arg)
{
        di_phys_t *pp = arg;
        tnode_t *pnp;
        tnode_t *ppnp;
        topo_faclist_t fl;
        topo_faclist_t *lp;
        int err;
        topo_led_state_t mode;
        topo_led_type_t type;
```

```
163         int err;
120         char *name, *slotname, *serial;

165         if (strcmp(topo_node_name(np), BAY) != 0)
166                 return;
122         if (strcmp(topo_node_name(np), DISK) != 0)
123                 return (TOPO_WALK_NEXT);

168         if (topo_node_facility(hp, np, TOPO_FAC_TYPE_INDICATOR,
125         if (topo_prop_get_string(np, TOPO_PGROUP_STORAGE,
126             TOPO_STORAGE_LOGICAL_DISK_NAME, &name, &err) != 0) {
127                 return (TOPO_WALK_NEXT);
128         }

130         if (strcmp(name, pp->dp_dev) != 0)
131                 return (TOPO_WALK_NEXT);

133         if (topo_prop_get_string(np, TOPO_PGROUP_STORAGE,
134             TOPO_STORAGE_SERIAL_NUM, &serial, &err) == 0) {
135                 pp->dp_serial = serial;
136         }

138         pnp = topo_node_parent(np);
139         ppnp = topo_node_parent(pnp);
140         if (strcmp(topo_node_name(pnp), BAY) == 0) {
141                 if (topo_node_facility(hp, pnp, TOPO_FAC_TYPE_INDICATOR,
169             TOPO_FAC_TYPE_ANY, &fl, &err) == 0) {
170                 for (lp = topo_list_next(&fl.tf_list); lp != NULL;
171                     lp = topo_list_next(lp)) {
172                         uint32_t prop;

174                         if (topo_prop_get_uint32(lp->tf_node,
175                             TOPO_PGROUP_FACILITY, TOPO_FACILITY_TYPE,
176                             &prop, &err) != 0) {
177                                 continue;
178                         }
179                         type = (topo_led_type_t)prop;

181                         if (topo_prop_get_uint32(lp->tf_node,
182                             TOPO_PGROUP_FACILITY, TOPO_LED_MODE,
183                             &prop, &err) != 0) {
184                                 continue;
185                         }
186                         mode = (topo_led_state_t)prop;

188                         switch (type) {
189                         case TOPO_LED_TYPE_SERVICE:
190                                 dip->dp_faulty = mode ? 1 : 0;
163                                         pp->dp_faulty = mode ? 1 : 0;
191                                 break;
192                         case TOPO_LED_TYPE_LOCATE:
193                                 dip->dp_locate = mode ? 1 : 0;
166                                         pp->dp_locate = mode ? 1 : 0;
194                                 break;
195                         default:
196                                 break;
197                         }
198                 }
199         }

201         if (topo_prop_get_string(np, TOPO_PGROUP_PROTOCOL,
202             TOPO_PROP_LABEL, &dip->dp_slotname, &err) == 0) {
203                 dip->dp_slotname = safe_strdup(dip->dp_slotname);
174                 if (topo_prop_get_string(pnp, TOPO_PGROUP_PROTOCOL,
175                     TOPO_PROP_LABEL, &slotname, &err) == 0) {
176                         pp->dp_slotname = slotname;
```



```
204         }

206         dip->dp_slot = topo_node_instance(np);
207         dip->dp_chassis = topo_node_instance(topo_node_parent(np));
208 }

210 static int
211 bay_walker(topo_hdl_t *hp, tnode_t *np, void *arg)
212 {
213         di_phys_t *dip;
214         int slot, chassis;

216         if (strcmp(topo_node_name(np), BAY) != 0)
217                 return (TOPO_WALK_NEXT);

219         slot = topo_node_instance(np);
220         chassis = topo_node_instance(topo_node_parent(np));

222         for (dip = list_head(&g_disks); dip != NULL;
223             dip = list_next(&g_disks, dip)) {
224                 if (dip->dp_slot == slot && dip->dp_chassis == chassis)
225                         return (TOPO_WALK_NEXT);
179                 pp->dp_slot = topo_node_instance(pnp);
226         }

228         dip = safe_zmalloc(sizeof (di_phys_t));
229         set_disk_bay_info(hp, np, dip);
230         list_insert_tail(&g_disks, dip);
231         return (TOPO_WALK_NEXT);
232 }
182         pp->dp_chassis = topo_node_instance(ppnp);

234 static int
235 disk_walker(topo_hdl_t *hp, tnode_t *np, void *arg)
236 {
237         char *dev;
238         di_phys_t *dip;
239         int err;

241         if (strcmp(topo_node_name(np), DISK) != 0)
242                 return (TOPO_WALK_NEXT);

244         if (topo_prop_get_string(np, TOPO_PGROUP_STORAGE,
245             TOPO_STORAGE_LOGICAL_DISK_NAME, &dev, &err) != 0) {
246                 return (TOPO_WALK_NEXT);
247         }

249         for (dip = list_head(&g_disks); dip != NULL;
250             dip = list_next(&g_disks, dip)) {
251                 if (strcmp(dip->dp_dev, dev) == 0) {
252                         if (topo_prop_get_string(np, TOPO_PGROUP_STORAGE,
253                             TOPO_STORAGE_SERIAL_NUM,
254                             &dip->dp_serial, &err) == 0) {
255                                 dip->dp_serial = safe_strdup(dip->dp_serial);
256                         }
257                         set_disk_bay_info(hp, topo_node_parent(np), dip);
258                 }
259         }
260         return (TOPO_WALK_NEXT);
184         return (TOPO_WALK_TERMINATE);
261 }

263 static void
264 walk_topo_snapshot(topo_hdl_t *hp, topo_walk_cb_t cb)
188 populate_physical(topo_hdl_t *hp, di_phys_t *pp)
265 {
```

```
266         int err = 0;
190         int err;
267         topo_walk_t *wp;

269         if ((wp = topo_walk_init(hp, FM_FMRI_SCHEME_HC, cb, NULL,
270             &err)) == NULL) {
193         pp->dp_faulty = pp->dp_locate = -1;
194         pp->dp_chassis = pp->dp_slot = -1;

196         err = 0;
197         wp = topo_walk_init(hp, FM_FMRI_SCHEME_HC, disk_walker, pp, &err);
198         if (wp == NULL) {
271                 fatal(-1, "unable to initialise topo walker: %s",
272                     topo_strerror(err));
273         }

275         while ((err = topo_walk_step(wp, TOPO_WALK_CHILD)) == TOPO_WALK_NEXT)
276                 ;

278         if (err == TOPO_WALK_ERR)
279                 fatal(-1, "topo walk failed");

280         topo_walk_fini(wp);
281 }

283 static void
284 enumerate_disks()
213 enumerate_disks(di_opts_t *opts)
285 {
286         topo_hdl_t *hp;
287         dm_descriptor_t *media;
217         int err, i;
288         int filter[] = { DM_DT_FIXED, -1 };
289         dm_descriptor_t *disk, *controller;
290         nvlist_t *mattrs, *dattrs, *cattrs;
220         nvlist_t *mattrs, *dattrs, *cattrs = NULL;

292         char *s, *c;
293         di_phys_t *dip;
222         uint64_t size, total;
223         uint32_t blocksize;
224         double total_in_GiB;
225         char sizestr[32];
226         char slotname[32];
227         char statestr[8];

229         char *vid, *pid, *opath, *c, *ctype = NULL;
230         boolean_t removable;
231         boolean_t ssd;
232         char device[MAXPATHLEN];
233         di_phys_t phys;
294         size_t len;
295         int err, i;

297         list_create(&g_disks, sizeof (di_phys_t), offsetof(di_phys_t, dp_next));

299         err = 0;
300         if ((media = dm_get_descriptors(DM_MEDIA, filter, &err)) == NULL) {
301                 fatal(-1, "failed to obtain media descriptors: %s\n",
302                     strerror(err));
303         }

242         err = 0;
243         hp = topo_open(TOPO_VERSION, NULL, &err);
244         if (hp == NULL) {
245                 fatal(-1, "unable to obtain topo handle: %s",
```



```
246                     topo_strerror(err));
247         }

249         err = 0;
250         (void) topo_snap_hold(hp, NULL, &err);
251         if (err != 0) {
252                 fatal(-1, "unable to hold topo snapshot: %s",
253                     topo_strerror(err));
254         }

305         for (i = 0; media != NULL && media[i] != NULL; i++) {
306                 if ((disk = dm_get_associated_descriptors(media[i],
307                     DM_DRIVE, &err)) == NULL) {
308                         continue;
309                 }

311                 dip = safe_zmalloc(sizeof (di_phys_t));

313                 mattrs = dm_get_attributes(media[i], &err);
314                 err = nvlist_lookup_uint64(mattrs, DM_SIZE, &dip->dp_size);
263                 err = nvlist_lookup_uint64(mattrs, DM_SIZE, &size);
315                 assert(err == 0);
316                 err = nvlist_lookup_uint32(mattrs, DM_BLOCKSIZE,
317                     &dip->dp_blksize);
265                 err = nvlist_lookup_uint32(mattrs, DM_BLOCKSIZE, &blocksize);
318                 assert(err == 0);
319                 nvlist_free(mattrs);

321                 dattrs = dm_get_attributes(disk[0], &err);

323                 nvlist_query_string(dattrs, DM_VENDOR_ID, &dip->dp_vid);
324                 nvlist_query_string(dattrs, DM_PRODUCT_ID, &dip->dp_pid);
325                 nvlist_query_string(dattrs, DM_OPATH, &dip->dp_dev);
271                 nvlist_query_string(dattrs, DM_VENDOR_ID, &vid);
272                 nvlist_query_string(dattrs, DM_PRODUCT_ID, &pid);
273                 nvlist_query_string(dattrs, DM_OPATH, &opath);

327                 dip->dp_vid = safe_strdup(dip->dp_vid);
328                 dip->dp_pid = safe_strdup(dip->dp_pid);
329                 dip->dp_dev = safe_strdup(dip->dp_dev);

331                 dip->dp_removable = B_FALSE;
275                 removable = B_FALSE;
332                 if (nvlist_lookup_boolean(dattrs, DM_REMOVABLE) == 0)
333                         dip->dp_removable = B_TRUE;
277                         removable = B_TRUE;

335                 dip->dp_ssd = B_FALSE;
279                 ssd = B_FALSE;
336                 if (nvlist_lookup_boolean(dattrs, DM_SOLIDSTATE) == 0)
337                         dip->dp_ssd = B_TRUE;
281                         ssd = B_TRUE;

339                 nvlist_free(dattrs);

341                 if ((controller = dm_get_associated_descriptors(disk[0],
342                     DM_CONTROLLER, &err)) != NULL) {
343                         cattrs = dm_get_attributes(controller[0], &err);
344                         nvlist_query_string(cattrs, DM_CTYPE, &dip->dp_ctype);
345                         dip->dp_ctype = safe_strdup(dip->dp_ctype);
346                         for (c = dip->dp_ctype; *c != '\0'; c++)
286                         nvlist_query_string(cattrs, DM_CTYPE, &ctype);
287                         ctype = strdup(ctype);
288                         for (c = ctype; *c != '\0'; c++)
347                                 *c = toupper(*c);
348                         nvlist_free(cattrs);
```

```
349                 }

351                 dm_free_descriptors(controller);
352                 dm_free_descriptors(disk);

354                 /*
355                  * Parse full device path to only show the device name,
356                  * i.e. c0t1d0.  Many paths will reference a particular
357                  * slice (c0t1d0s0), so remove the slice if present.
358                  */
359                 if ((c = strrchr(dip->dp_dev, '/')) != NULL) {
360                         s = dip->dp_dev;
361                         while ((*s++ = *++c))
362                                 ;
363                 }
364                 len = strlen(dip->dp_dev);
365                 if (dip->dp_dev[len - 2] == 's' &&
366                     dip->dp_dev[len - 1] >= '0' &&
367                     dip->dp_dev[len - 1] <= '9')
368                         dip->dp_dev[len - 2] = '\0';
297                 if ((c = strrchr(opath, '/')) != NULL)
298                         (void) strlcpy(device, c + 1, sizeof (device));
299                 else
300                         (void) strlcpy(device, opath, sizeof (device));
301                 len = strlen(device);
302                 if (device[len - 2] == 's' &&
303                     (device[len - 1] >= '0' && device[len - 1] <= '9'))
304                         device[len - 2] = '\0';

370                 dip->dp_faulty = dip->dp_locate = -1;
371                 dip->dp_chassis = dip->dp_slot = -1;
372                 list_insert_tail(&g_disks, dip);
373         }
306                 bzero(&phys, sizeof (phys));
307                 phys.dp_dev = device;
308                 populate_physical(hp, &phys);

375         dm_free_descriptors(media);

377         /*
378          * Walk toplogy information to populate serial, chassis,
379          * slot, faulty and locator information.
380          */

382         err = 0;
383         hp = topo_open(TOPO_VERSION, NULL, &err);
384         if (hp == NULL) {
385                 fatal(-1, "unable to obtain topo handle: %s",
386                     topo_strerror(err));
387         }

389         err = 0;
390         (void) topo_snap_hold(hp, NULL, &err);
391         if (err != 0) {
392                 fatal(-1, "unable to hold topo snapshot: %s",
393                     topo_strerror(err));
394         }

396         walk_topo_snapshot(hp, disk_walker);

398         if (g_opts.di_allslots)
399                 walk_topo_snapshot(hp, bay_walker);

401         topo_snap_release(hp);
402         topo_close(hp);
403 }
```



```
405 static void
406 show_disks()
407 {
408         uint64_t total;
409         double total_in_GiB;
410         char *sizestr = NULL, *slotname = NULL, *statestr = NULL;
411         di_phys_t *dip;

413         for (dip = list_head(&g_disks); dip != NULL;
414             dip = list_next(&g_disks, dip)) {
415                 /*
416                  * The size is given in blocks, so multiply the number
417                  * of blocks by the block size to get the total size,
418                  * then convert to GiB.
419                  */
420                 total = dip->dp_size * dip->dp_blksize;
315                 total = size * blocksize;

422                 if (g_opts.di_parseable) {
423                         (void) safe_asprintf(&sizestr, "%llu", total);
317                 if (opts->di_parseable) {
318                         (void) snprintf(sizestr, sizeof (sizestr),
319                             "%llu", total);
424                 } else {
425                         total_in_GiB = (double)total /
426                             1024.0 / 1024.0 / 1024.0;
427                         (void) safe_asprintf(&sizestr,
428                             "%7.2f GiB", (total_in_GiB));
323                         (void) snprintf(sizestr, sizeof (sizestr),
324                             "%7.2f GiB", total_in_GiB);
429                 }

431                 if (g_opts.di_parseable) {
432                         (void) safe_asprintf(&slotname, "%d,%d",
433                             dip->dp_chassis, dip->dp_slot);
434                 } else if (dip->dp_slotname != NULL) {
435                         (void) safe_asprintf(&slotname, "[%d] %s",
436                             dip->dp_chassis, dip->dp_slotname);
327                 if (opts->di_parseable) {
328                         (void) snprintf(slotname, sizeof (slotname), "%d,%d",
329                             phys.dp_chassis, phys.dp_slot);
330                 } else if (phys.dp_slotname != NULL) {
331                         (void) snprintf(slotname, sizeof (slotname),
332                             "[%d] %s", phys.dp_chassis, phys.dp_slotname);
437                 } else {
438                         slotname = safe_strdup("-");
334                         slotname[0] = '-';
335                         slotname[1] = '\0';
439                 }

441                 if (g_opts.di_condensed) {
442                         (void) safe_asprintf(&statestr, "%c%c%c%c",
443                             condensed_tristate(dip->dp_faulty, 'F'),
444                             condensed_tristate(dip->dp_locate, 'L'),
445                             condensed_tristate(dip->dp_removable, 'R'),
446                             condensed_tristate(dip->dp_ssd, 'S'));
338                 if (opts->di_condensed) {
339                         (void) snprintf(statestr, sizeof (statestr), "%c%c%c%c",
340                             condensed_tristate(phys.dp_faulty, 'F'),
341                             condensed_tristate(phys.dp_locate, 'L'),
342                             condensed_tristate(removable, 'R'),
343                             condensed_tristate(ssd, 'S'));
447                 }

449                 if (g_opts.di_physical) {
```

```
                        if (g_opts.di_scripted) {
                if (opts->di_physical) {
                        if (opts->di_scripted) {
                                printf("%s\t%s\t%s\t%s\t%s\t%s\t%s\n",
                                    display_string(dip->dp_dev),
                                    display_string(dip->dp_vid),
                                    display_string(dip->dp_pid),
                                    display_string(dip->dp_serial),
                                    display_tristate(dip->dp_faulty),
                                    display_tristate(dip->dp_locate), slotname);
                                    device, vid, pid,
                                    display_string(phys.dp_serial),
                                    display_tristate(phys.dp_faulty),
                                    display_tristate(phys.dp_locate), slotname);
                        } else {
                                printf("%-22s  %-8s %-16s "
                                    "%-20s %-3s %-3s %s\n",
                                    display_string(dip->dp_dev),
                                    display_string(dip->dp_vid),
                                    display_string(dip->dp_pid),
                                    display_string(dip->dp_serial),
                                    display_tristate(dip->dp_faulty),
                                    display_tristate(dip->dp_locate), slotname);
                                    device, vid, pid,
                                    display_string(phys.dp_serial),
                                    display_tristate(phys.dp_faulty),
                                    display_tristate(phys.dp_locate), slotname);
                        }
                } else if (g_opts.di_condensed) {
                        if (g_opts.di_scripted) {
                } else if (opts->di_condensed) {
                        if (opts->di_scripted) {
                                printf("%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\n",
                                    display_string(dip->dp_ctype),
                                    display_string(dip->dp_dev),
                                    display_string(dip->dp_vid),
                                    display_string(dip->dp_pid),
                                    display_string(dip->dp_serial),
                                    ctype, device, vid, pid,
                                    display_string(phys.dp_serial),
                                    sizestr, statestr, slotname);
                        } else {
                                printf("%-7s %-22s  %-8s %-16s "
                                    "%-20s\n\t%-13s %-4s %s\n",
                                    display_string(dip->dp_ctype),
                                    display_string(dip->dp_dev),
                                    display_string(dip->dp_vid),
                                    display_string(dip->dp_pid),
                                    display_string(dip->dp_serial),
                                    ctype, device, vid, pid,
                                    display_string(phys.dp_serial),
                                    sizestr, statestr, slotname);
                        }
                } else {
                        if (g_opts.di_scripted) {
                        if (opts->di_scripted) {
                                printf("%s\t%s\t%s\t%s\t%s\t%s\t%s\n",
                                    display_string(dip->dp_ctype),
                                    display_string(dip->dp_dev),
                                    display_string(dip->dp_vid),
                                    display_string(dip->dp_pid), sizestr,
                                    display_tristate(dip->dp_removable),
                                    display_tristate(dip->dp_ssd));
                                    ctype, device, vid, pid, sizestr,
                                    display_tristate(removable),
                                    display_tristate(ssd));
```



```
                        } else {
                                printf("%-7s %-22s  %-8s %-16s "
                                    "%-13s %-3s %-3s\n",
                                    display_string(dip->dp_ctype),
                                    display_string(dip->dp_dev),
                                    display_string(dip->dp_vid),
                                    display_string(dip->dp_pid), sizestr,
                                    display_tristate(dip->dp_removable),
                                    display_tristate(dip->dp_ssd));
                                    "%-13s %-3s %-3s\n", ctype, device,
                                    vid, pid, sizestr,
                                    display_tristate(removable),
                                    display_tristate(ssd));
                        }
                }
        free(sizestr); free(slotname); free(statestr);
        sizestr = slotname = statestr = NULL;
        }
}

static void
cleanup()
{
        di_phys_t *dip;
        while ((dip = list_head(&g_disks)) != NULL) {
                list_remove(&g_disks, dip);
                free(dip->dp_vid);
                free(dip->dp_pid);
                free(dip->dp_dev);
                free(dip->dp_ctype);
                free(dip->dp_serial);
                free(dip->dp_slotname);
                free(dip);
                free(ctype);
                nvlist_free(cattrs);
                nvlist_free(dattrs);
                dm_free_descriptors(controller);
                dm_free_descriptors(disk);
        }
        list_destroy(&g_disks);

        dm_free_descriptors(media);
        topo_snap_release(hp);
        topo_close(hp);
}

int
main(int argc, char *argv[])
{
        char c;

        while ((c = getopt(argc, argv, ":acHPp")) != EOF) {
        di_opts_t opts = {
                .di_condensed = B_FALSE,
                .di_scripted = B_FALSE,
                .di_physical = B_FALSE,
                .di_parseable = B_FALSE
        };

        while ((c = getopt(argc, argv, ":cHPp")) != EOF) {
                switch (c) {
                case 'a':
                        g_opts.di_allslots = B_TRUE;
                        break;
                case 'c':
                        g_opts.di_condensed = B_TRUE;
```

```
416                         if (opts.di_physical) {
417                                 usage(argv[0]);
418                                 fatal(1, "-c and -P are mutually exclusive\n");
419                         }
420                         opts.di_condensed = B_TRUE;
541                         break;
542                 case 'H':
543                         g_opts.di_scripted = B_TRUE;
423                         opts.di_scripted = B_TRUE;
544                         break;
545                 case 'P':
546                         g_opts.di_physical = B_TRUE;
426                         if (opts.di_condensed) {
427                                 usage(argv[0]);
428                                 fatal(1, "-c and -P are mutually exclusive\n");
429                         }
430                         opts.di_physical = B_TRUE;
547                         break;
548                 case 'p':
549                         g_opts.di_parseable = B_TRUE;
433                         opts.di_parseable = B_TRUE;
550                         break;
551                 case '?':
552                         usage(argv[0]);
553                         fatal(1, "unknown option -%c\n", optopt);
554                 default:
555                         fatal(-1, "unexpected error on option -%c\n", optopt);
556                 }
557         }

559         if (g_opts.di_condensed && g_opts.di_physical) {
560                 usage(argv[0]);
561                 fatal(1, "-c and -P are mutually exclusive\n");
562         }

564         if (!g_opts.di_scripted) {
565                 if (g_opts.di_physical) {
443         if (!opts.di_scripted) {
444                 if (opts.di_physical) {
566                         printf("DISK                    VID      PID"
567                             "              SERIAL               FLT LOC"
568                             " LOCATION\n");
569                 } else if (g_opts.di_condensed) {
448                 } else if (opts.di_condensed) {
570                         printf("TYPE    DISK                    VID      PID"
571                             "              SERIAL\n");
572                         printf("\tSIZE          FLRS LOCATION\n");
573                 } else {
574                         printf("TYPE    DISK                    VID      PID"
575                             "              SIZE          RMV SSD\n");
576                 }
577         }

579         enumerate_disks();
580         show_disks();
581         cleanup();
458         enumerate_disks(&opts);

583         return (0);
584 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    5852 Sun Oct  8 15:35:27 2017
new/usr/src/man/man1m/diskinfo.1m
8708 Want diskinfo(1m) to list all disk bays, including those don't have disk in
**********************************************************
.\"
.\" This file and its contents are supplied under the terms of the
.\" Common Development and Distribution License ("CDDL"), version 1.0.
.\" You may only use this file in accordance with the terms of version
.\" 1.0 of the CDDL.
.\"
.\" A full copy of the text of the CDDL should have accompanied this
.\" source.  A copy of the CDDL is also available via the Internet at
.\" http://www.illumos.org/license/CDDL.
.\"
.\" Copyright 2014 Joyent, Inc.
.\" Copyright 2016 Nexenta Systems, Inc.
.\"
.Dd October 06, 2017
.Dd July 20, 2016
.Dt DISKINFO 1M
.Os
.Sh NAME
.Nm diskinfo
.Nd provide disk device inventory and status
.Sh SYNOPSIS
.Nm
.Op Fl aHp
.Op Fl Hp
.Op Fl c Ns | Ns Fl P
.Sh DESCRIPTION
The diskinfo tool provides information about the disk devices in the system.
Because it interacts with the kernel's device management subsystem, this tool
can be used only from the global zone.
If run in any other zone, its output will be incomplete and unreliable.
.Pp
There are three main modes.
The default mode, when neither the
.Fl c
nor
.Fl P
option is specified, provides a basic inventory of the disk devices in the
system.
Each line describes a single device and contains the device's attachment bus or
fabric type, the base name of the device in the
.Pa /dev/dsk
directory, the disk's vendor and product identification strings, the size
.Pq storage capacity
of the device, whether the device is removable, and whether it is solid-state.
.Pp
The
.Fl P
option selects physical mode.
In this mode, each line of output likewise describes one disk device; however,
the fields provided indicate the base name of the device in the
.Pa /dev/dsk
directory, the disk's vendor and product identification strings, the serial
number of the device, whether the device is faulty as diagnosed by
.Xr fmd 1M ,
whether the locate or identification indicator is on for the device
.Pq if one is present ,
and the chassis and bay number containing the disk if known.
.Pp
The
.Fl c
option selects compact mode.
```



```
This mode provides all of the information provided by both the default mode and
physical mode in a compact format.
.Pp
See
.Sx OUTPUT FIELDS
below for a detailed description of each column.
.Sh OPTIONS
.Bl -tag -width Ds
.It Fl a
Show all disk bays, including those don't have disks installed.
.It Fl c
Select compact mode output.
At most one of
.Fl c
and
.Fl P
may be present on the command line.
.It Fl H
Do not print a header.
This provides output suitable for passing into text processing tools.
.It Fl P
Select physical mode output.
At most one of
.Fl P
and
.Fl c
may be present on the command line.
.It Fl p
Parsable output.
When
.Fl p
is selected, the size
.Pq storage capacity
is output in bytes instead of in human-readable units, and the device's location
.Pq if known
is provided as a comma-separated chassis and bay number instead of a
human-readable location.
This option may be used in any output mode and is intended for use by scripts or
other robotic tooling.
.El
.Sh OUTPUT FIELDS
.Bl -tag -width "LOCATION"
.It Sy DISK
The base name of the device node within the
.Pa /dev/dsk
directory.
The names of partitions and/or slices, if any, are derived from this name as
described by
.Xr prtvtoc 1M .
.Pp
This field is available in all output modes.
.It Sy FLRS
A condensed field incorporating the same information as the
.Sy FLT , LOC , RMV ,
and
.Sy SSD
fields.
Each field is condensed to a single character.
If the field is true, the first letter of the field name will appear in its
position in the string; otherwise, the
.Qq Sy -
character will appear instead.
.Pp
This field is available only in compact output mode.
.It Sy FLT
A boolean field indicating whether the device is faulty; specifically, whether
```

```
126 the fault indicator
127 .Pq if one is present
128 is active.
129 .Pp
130 This field is available only in physical output mode.
131 .It Sy LOC
132 A boolean field indicating whether the locate or identify indicator, if any,
133 associated with the device’s bay, is active.
134 .Pp
135 This field is available only in physical output mode.
136 .It Sy LOCATION
137 The physical chassis and bay name
138 .Po or chassis and bay numbers, if
139 .Fl p
140 is given
141 .Pc
142 in which the device is located.
143 The chassis number is identified in human-readable output within
144 .Bq square brackets ;
145 chassis 0 is the host chassis itself.
146 The bay name, if any, is provided by the enclosure, typically via a SCSI
147 Enclosure Services processor.
148 .Pp
149 This field is available in compact and physical output modes.
150 .It Sy PID
151 The product identification string reported by the device.
152 .Pp
153 This field is available in all output modes.
154 .It Sy RMV
155 A boolean field indicating whether the device is removable.
156 USB storage devices, most optical drives and changers, and certain other devices
157 that report themselves as removable will be identified as such.
158 .Pp
159 This field is available only in default output mode.
160 .It Sy SERIAL
161 The serial number of the device.
162 The entire serial number is reported if the device and its drivers provide it.
163 .Pp
164 This field is available in compact and physical output modes.
165 .It Sy SIZE
166 The device’s storage capacity.
167 If the
168 .Fl p
169 option is given, this is reported in bytes; otherwise, it is reported in a
170 human-readable format with units specified.
171 All units are based on powers of 2 and are expressed in SI standard notation.
172 .Pp
173 This field is available in compact and default output modes.
174 .It Sy SSD
175 A boolean field indicating whether the device is solid-state.
176 .Pp
177 This field is available only in default output mode.
178 .It Sy TYPE
179 The transport
180 .Pq fabric or bus
181 type by which the storage device is attached to the host, if known.
182 Typical transports include SCSI and USB.
183 .Pp
184 This field is available in compact and default output modes.
185 .It Sy VID
186 The vendor identification string reported by the device.
187 .Pp
188 This field is available in all output modes.
189 .El
190 .Sh SEE ALSO
191 .Xr fmd 1M ,
```



```
192 .Xr prtvtoc 1M ,
193 .Xr sd 7D
```