```
**********************************************************
   47608 Thu Sep  5 23:02:02 2013
new/usr/src/cmd/fm/fmadm/common/faulty.c
3373 gcc >= 4.5 concerns about offsetof()
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
 */

#include <sys/types.h>
#include <fmadm.h>
#include <errno.h>
#include <limits.h>
#include <strings.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <fm/fmd_log.h>
#include <sys/fm/protocol.h>
#include <fm/libtopo.h>
#include <fm/fmd_adm.h>
#include <fm/fmd_msg.h>
#include <dlfcn.h>
#include <sys/systeminfo.h>
#include <sys/utsname.h>
#include <libintl.h>
#include <locale.h>
#include <sys/smbios.h>
#include <libdevinfo.h>
#include <stdlib.h>

#if defined(__GNUC__)
#define offsetof(s, m)  __builtin_offsetof(s, m)
#else
#endif /* ! codereview */
#define offsetof(s, m)  ((size_t)(&(((s *)0)->m)))
#endif
#endif /* ! codereview */

/*
 * Fault records are added to catalog by calling add_fault_record_to_catalog()
 * records are stored in order of importance to the system.
 * If -g flag is set or not_suppressed is not set and the class fru, fault,
 * type are the same then details are merged into an existing record, with uuid
```



```
 * records are stored in time order.
 * For each record information is extracted from nvlist and merged into linked
 * list each is checked for identical records for which percentage certainty are
 * added together.
 * print_catalog() is called to print out catalog and release external resources
 *
 *                          /---------------\
 *      status_rec_list -> |                | -|
 *                          \---------------/
 *                                 \/
 *                          /---------------\    /-------\    /-------\
 *      status_fru_list     | status_record | -> | uurec | -> | uurec | -|
 *            \/            |                | |- |       | <- |       |
 *      /-------------\     |                |    \-------/    \-------/
 *      |             | -> |                |       \/           \/
 *      \-------------/     |                |    /-------\    /-------\
 *            \/            |                | -> | asru  | -> | asru  |
 *            ---           |                |    |       | <- |       |
 *                          |                |    \-------/    \-------/
 *      status_asru_list    |  class         |
 *            \/            |  resource      |    /-------\    /-------\
 *      /-------------\     |  fru           | -> | list  | -> | list  |
 *      |             | -> |  serial        |    |       | <- |       |
 *      \-------------/     |                |    \-------/    \-------/
 *            \/            \---------------/
 *            ---               \/    /\
 *                          /---------------\
 *                          | status_record |
 *                          \---------------/
 *
 * Fmadm faulty takes a number of options which affect the format of the
 * output displayed. By default, the display reports the FRU and ASRU along
 * with other information on per-case basis as in the example below.
 *
 * -------------- ------------------------------------  -------------- -------
 * TIME           EVENT-ID                              MSG-ID         SEVERITY
 * -------------- ------------------------------------  -------------- -------
 * Sep 21 10:01:36 d482f935-5c8f-e9ab-9f25-d0aaafec1e6c  AMD-8000-2F    Major
 *
 * Fault class  : fault.memory.dimm_sb
 * Affects      : mem:///motherboard=0/chip=0/memory-controller=0/dimm=0/rank=0
 *                  faulted but still in service
 * FRU          : "CPU 0 DIMM 0" (hc://.../memory-controller=0/dimm=0)
 *                  faulty
 *
 * Description  : The number of errors associated with this memory module has
 *                exceeded acceptable levels.  Refer to
 *                http://illumos.org/msg/AMD-8000-2F for more information.
 *
 * Response     : Pages of memory associated with this memory module are being
 *                removed from service as errors are reported.
 *
 * Impact       : Total system memory capacity will be reduced as pages are
 *                retired.
 *
 * Action       : Schedule a repair procedure to replace the affected memory
 *                module.  Use fmdump -v -u <EVENT_ID> to identify the module.
 *
 * The -v flag is similar, but adds some additonal information such as the
 * resource. The -s flag is also similar but just gives the top line summary.
 * All these options (ie without the -f or -r flags) use the print_catalog()
 * function to do the display.
 *
 * The -f flag changes the output so that it appears sorted on a per-fru basis.
 * The output is somewhat cut down compared to the default output. If -f is
 * used, then print_fru() is used to print the output.
```

```
 *
 * --------------------------------------------------------------------------
 * "SLOT 2" (hc://.../hostbridge=3/pciexrc=3/pciexbus=4/pciexdev=0) faulty
 * 5ca4aeb3-36...f6be-c2e8166dc484 2 suspects in this FRU total certainty 100%
 *
 * Description  : A problem was detected for a PCI device.
 *              Refer to http://illumos.org/msg/PCI-8000-7J
 *              for more information.
 *
 * Response     : One or more device instances may be disabled
 *
 * Impact       : Possible loss of services provided by the device instances
 *              associated with this fault
 *
 * Action       : Schedule a repair procedure to replace the affected device.
 *              Use fmdump -v -u <EVENT_ID> to identify the device or contact
 *              Sun for support.
 *
 * The -r flag changes the output so that it appears sorted on a per-asru basis.
 * The output is very much cut down compared to the default output, just giving
 * the asru fmri and state. Here print_asru() is used to print the output.
 *
 * mem:///motherboard=0/chip=0/memory-controller=0/dimm=0/rank=0        degraded
 *
 * For all fmadm faulty options, the sequence of events is
 *
 * 1) Walk through all the cases in the system using fmd_adm_case_iter() and
 * for each case call dfault_rec(). This will call add_fault_record_to_catalog()
 * This will extract the data from the nvlist and call catalog_new_record() to
 * save the data away in various linked lists in the catalogue.
 *
 * 2) Once this is done, the data can be supplemented by using
 * fmd_adm_rsrc_iter(). However this is now only necessary for the -i option.
 *
 * 3) Finally print_catalog(), print_fru() or print_asru() are called as
 * appropriate to display the information from the catalogue sorted in the
 * requested way.
 *
 */

typedef struct name_list {
        struct name_list *next;
        struct name_list *prev;
        char *name;
        uint8_t pct;
        uint8_t max_pct;
        ushort_t count;
        int status;
        char *label;
} name_list_t;

typedef struct ari_list {
        char *ari_uuid;
        struct ari_list *next;
} ari_list_t;

typedef struct uurec {
        struct uurec *next;
        struct uurec *prev;
        char *uuid;
        ari_list_t *ari_uuid_list;
        name_list_t *asru;
        uint64_t sec;
        nvlist_t *event;
} uurec_t;
```



```
typedef struct uurec_select {
        struct uurec_select *next;
        char *uuid;
} uurec_select_t;

typedef struct host_id {
        char *chassis;
        char *server;
        char *platform;
        char *domain;
        char *product_sn;
} hostid_t;

typedef struct host_id_list {
        hostid_t hostid;
        struct host_id_list *next;
} host_id_list_t;

typedef struct status_record {
        hostid_t *host;
        int nrecs;
        uurec_t *uurec;
        char *severity;                 /* in C locale */
        char *msgid;
        name_list_t *class;
        name_list_t *resource;
        name_list_t *asru;
        name_list_t *fru;
        name_list_t *serial;
        uint8_t not_suppressed;
        uint8_t injected;
} status_record_t;

typedef struct sr_list {
        struct sr_list *next;
        struct sr_list *prev;
        struct status_record *status_record;
} sr_list_t;

typedef struct resource_list {
        struct resource_list *next;
        struct resource_list *prev;
        sr_list_t *status_rec_list;
        char *resource;
        uint8_t not_suppressed;
        uint8_t injected;
        uint8_t max_pct;
} resource_list_t;

sr_list_t *status_rec_list;
resource_list_t *status_fru_list;
resource_list_t *status_asru_list;

static int max_display;
static int max_fault = 0;
static topo_hdl_t *topo_handle;
static host_id_list_t *host_list;
static int n_server;
static int opt_g;
static fmd_msg_hdl_t *fmadm_msghdl = NULL; /* handle for libfmd_msg calls */

static char *
format_date(char *buf, size_t len, uint64_t sec)
{
        if (sec > LONG_MAX) {
                (void) fprintf(stderr,
```

```
260                     "record time is too large for 32-bit utility\n");
261                 (void) snprintf(buf, len, "0x%llx", sec);
262         } else {
263                 time_t tod = (time_t)sec;
264                 time_t now = time(NULL);
265                 if (tod > now+60 ||
266                     tod < now - 6L*30L*24L*60L*60L) { /* 6 months ago */
267                         (void) strftime(buf, len, "%b %d %Y    ",
268                             localtime(&tod));
269                 } else {
270                         (void) strftime(buf, len, "%b %d %T", localtime(&tod));
271                 }
272         }

274         return (buf);
275 }

277 static hostid_t *
278 find_hostid_in_list(char *platform, char *chassis, char *server, char *domain,
279     char *product_sn)
280 {
281         hostid_t *rt = NULL;
282         host_id_list_t *hostp;

284         if (platform == NULL)
285                 platform = "-";
286         if (server == NULL)
287                 server = "-";
288         hostp = host_list;
289         while (hostp) {
290                 if (hostp->hostid.platform &&
291                     strcmp(hostp->hostid.platform, platform) == 0 &&
292                     hostp->hostid.server &&
293                     strcmp(hostp->hostid.server, server) == 0 &&
294                     (chassis == NULL || hostp->hostid.chassis == NULL ||
295                     strcmp(chassis, hostp->hostid.chassis) == 0) &&
296                     (product_sn == NULL || hostp->hostid.product_sn == NULL ||
297                     strcmp(product_sn, hostp->hostid.product_sn) == 0) &&
298                     (domain == NULL || hostp->hostid.domain == NULL ||
299                     strcmp(domain, hostp->hostid.domain) == 0)) {
300                         rt = &hostp->hostid;
301                         break;
302                 }
303                 hostp = hostp->next;
304         }
305         if (rt == NULL) {
306                 hostp = malloc(sizeof (host_id_list_t));
307                 hostp->hostid.platform = strdup(platform);
308                 hostp->hostid.product_sn =
309                     product_sn ? strdup(product_sn) : NULL;
310                 hostp->hostid.server = strdup(server);
311                 hostp->hostid.chassis = chassis ? strdup(chassis) : NULL;
312                 hostp->hostid.domain = domain ? strdup(domain) : NULL;
313                 hostp->next = host_list;
314                 host_list = hostp;
315                 rt = &hostp->hostid;
316                 n_server++;
317         }
318         return (rt);
319 }

321 static hostid_t *
322 find_hostid(nvlist_t *nvl)
323 {
324         char *platform = NULL, *chassis = NULL, *server = NULL, *domain = NULL;
325         char *product_sn = NULL;
```



```
326         nvlist_t *auth, *fmri;
327         hostid_t *rt = NULL;

329         if (nvlist_lookup_nvlist(nvl, FM_SUSPECT_DE, &fmri) == 0 &&
330             nvlist_lookup_nvlist(fmri, FM_FMRI_AUTHORITY, &auth) == 0) {
331                 (void) nvlist_lookup_string(auth, FM_FMRI_AUTH_PRODUCT,
332                     &platform);
333                 (void) nvlist_lookup_string(auth, FM_FMRI_AUTH_PRODUCT_SN,
334                     &product_sn);
335                 (void) nvlist_lookup_string(auth, FM_FMRI_AUTH_SERVER, &server);
336                 (void) nvlist_lookup_string(auth, FM_FMRI_AUTH_CHASSIS,
337                     &chassis);
338                 (void) nvlist_lookup_string(auth, FM_FMRI_AUTH_DOMAIN, &domain);
339                 rt = find_hostid_in_list(platform, chassis, server,
340                     domain, product_sn);
341         }
342         return (rt);
343 }

345 static char *
346 get_nvl2str_topo(nvlist_t *nvl)
347 {
348         char *name = NULL;
349         char *tname;
350         int err;
351         char *scheme = NULL;
352         char *mod_name = NULL;
353         char buf[128];

355         if (topo_handle == NULL)
356                 topo_handle = topo_open(TOPO_VERSION, 0, &err);
357         if (topo_fmri_nvl2str(topo_handle, nvl, &tname, &err) == 0) {
358                 name = strdup(tname);
359                 topo_hdl_strfree(topo_handle, tname);
360         } else {
361                 (void) nvlist_lookup_string(nvl, FM_FMRI_SCHEME, &scheme);
362                 (void) nvlist_lookup_string(nvl, FM_FMRI_MOD_NAME, &mod_name);
363                 if (scheme && strcmp(scheme, FM_FMRI_SCHEME_FMD) == 0 &&
364                     mod_name) {
365                         (void) snprintf(buf, sizeof (buf), "%s:///module/%s",
366                             scheme, mod_name);
367                         name = strdup(buf);
368                 }
369         }
370         return (name);
371 }

373 static int
374 set_priority(char *s)
375 {
376         int rt = 0;

378         if (s) {
379                 if (strcmp(s, "Minor") == 0)
380                         rt = 1;
381                 else if (strcmp(s, "Major") == 0)
382                         rt = 10;
383                 else if (strcmp(s, "Critical") == 0)
384                         rt = 100;
385         }
386         return (rt);
387 }

389 static int
390 cmp_priority(char *s1, char *s2, uint64_t t1, uint64_t t2, uint8_t p1,
391     uint8_t p2)
```

```
392 {
393         int r1, r2;
394         int rt;

396         r1 = set_priority(s1);
397         r2 = set_priority(s2);
398         rt = r1 - r2;
399         if (rt == 0) {
400                 if (t1 > t2)
401                         rt = 1;
402                 else if (t1 < t2)
403                         rt = -1;
404                 else
405                         rt = p1 - p2;
406         }
407         return (rt);
408 }

410 /*
411  * merge two lists into one, by comparing enties in new and moving into list if
412  * name is not there or free off memory for names which are already there
413  * add_pct indicates if pct is the sum or highest pct
414  */
415 static name_list_t *
416 merge_name_list(name_list_t **list, name_list_t *new, int add_pct)
417 {
418         name_list_t *lp, *np, *sp, *rt = NULL;
419         int max_pct;

421         rt = *list;
422         np = new;
423         while (np) {
424                 lp = *list;
425                 while (lp) {
426                         if (strcmp(lp->name, np->name) == 0)
427                                 break;
428                         lp = lp->next;
429                         if (lp == *list)
430                                 lp = NULL;
431                 }
432                 if (np->next == new)
433                         sp = NULL;
434                 else
435                         sp = np->next;
436                 if (lp) {
437                         lp->status |= (np->status & FM_SUSPECT_FAULTY);
438                         if (add_pct) {
439                                 lp->pct += np->pct;
440                                 lp->count += np->count;
441                         } else if (np->pct > lp->pct) {
442                                 lp->pct = np->pct;
443                         }
444                         max_pct = np->max_pct;
445                         if (np->label)
446                                 free(np->label);
447                         free(np->name);
448                         free(np);
449                         np = NULL;
450                         if (max_pct > lp->max_pct) {
451                                 lp->max_pct = max_pct;
452                                 if (lp->max_pct > lp->prev->max_pct &&
453                                     lp != *list) {
454                                         lp->prev->next = lp->next;
455                                         lp->next->prev = lp->prev;
456                                         np = lp;
457                                 }
```



```
458                         }
459                 }
460                 if (np) {
461                         lp = *list;
462                         if (lp) {
463                                 if (np->max_pct > lp->max_pct) {
464                                         np->next = lp;
465                                         np->prev = lp->prev;
466                                         lp->prev->next = np;
467                                         lp->prev = np;
468                                         *list = np;
469                                         rt = np;
470                                 } else {
471                                         lp = lp->next;
472                                         while (lp != *list &&
473                                             np->max_pct < lp->max_pct) {
474                                                 lp = lp->next;
475                                         }
476                                         np->next = lp;
477                                         np->prev = lp->prev;
478                                         lp->prev->next = np;
479                                         lp->prev = np;
480                                 }
481                         } else {
482                                 *list = np;
483                                 np->next = np;
484                                 np->prev = np;
485                                 rt = np;
486                         }
487                 }
488                 np = sp;
489         }
490         return (rt);
491 }

493 static name_list_t *
494 alloc_name_list(char *name, uint8_t pct)
495 {
496         name_list_t *nlp;

498         nlp = malloc(sizeof (*nlp));
499         nlp->name = strdup(name);
500         nlp->pct = pct;
501         nlp->max_pct = pct;
502         nlp->count = 1;
503         nlp->next = nlp;
504         nlp->prev = nlp;
505         nlp->status = 0;
506         nlp->label = NULL;
507         return (nlp);
508 }

510 static status_record_t *
511 new_record_init(uurec_t *uurec_p, char *msgid, name_list_t *class,
512     name_list_t *fru, name_list_t *asru, name_list_t *resource,
513     name_list_t *serial, boolean_t not_suppressed,
514     hostid_t *hostid, boolean_t injected)
515 {
516         status_record_t *status_rec_p;

518         status_rec_p = (status_record_t *)malloc(sizeof (status_record_t));
519         status_rec_p->nrecs = 1;
520         status_rec_p->host = hostid;
521         status_rec_p->uurec = uurec_p;
522         uurec_p->next = NULL;
523         uurec_p->prev = NULL;
```

```
524         uurec_p->asru = asru;
525         if ((status_rec_p->severity = fmd_msg_getitem_id(fmadm_msghdl, NULL,
526             msgid, FMD_MSG_ITEM_SEVERITY)) == NULL)
527                 status_rec_p->severity = strdup("unknown");
528         status_rec_p->class = class;
529         status_rec_p->fru = fru;
530         status_rec_p->asru = asru;
531         status_rec_p->resource = resource;
532         status_rec_p->serial = serial;
533         status_rec_p->msgid = strdup(msgid);
534         status_rec_p->not_suppressed = not_suppressed;
535         status_rec_p->injected = injected;
536         return (status_rec_p);
537 }

539 /*
540  * add record to given list maintaining order higher priority first.
541  */
542 static void
543 add_rec_list(status_record_t *status_rec_p, sr_list_t **list_pp)
544 {
545         sr_list_t *tp, *np, *sp;
546         int order;
547         uint64_t sec;

549         np = malloc(sizeof (sr_list_t));
550         np->status_record = status_rec_p;
551         sec = status_rec_p->uurec->sec;
552         if ((sp = *list_pp) == NULL) {
553                 *list_pp = np;
554                 np->next = np;
555                 np->prev = np;
556         } else {
557                 /* insert new record in front of lower priority */
558                 tp = sp;
559                 order = cmp_priority(status_rec_p->severity,
560                     sp->status_record->severity, sec,
561                     tp->status_record->uurec->sec, 0, 0);
562                 if (order > 0) {
563                         *list_pp = np;
564                 } else {
565                         tp = sp->next;
566                         while (tp != sp &&
567                             cmp_priority(status_rec_p->severity,
568                             tp->status_record->severity, sec,
569                             tp->status_record->uurec->sec, 0, 0)) {
570                                 tp = tp->next;
571                         }
572                 }
573                 np->next = tp;
574                 np->prev = tp->prev;
575                 tp->prev->next = np;
576                 tp->prev = np;
577         }
578 }

580 static void
581 add_resource(status_record_t *status_rec_p, resource_list_t **rp,
582     resource_list_t *np)
583 {
584         int order;
585         uint64_t sec;
586         resource_list_t *sp, *tp;
587         status_record_t *srp;
588         char *severity = status_rec_p->severity;
```



```
590         add_rec_list(status_rec_p, &np->status_rec_list);
591         if ((sp = *rp) == NULL) {
592                 np->next = np;
593                 np->prev = np;
594                 *rp = np;
595         } else {
596                 /*
597                  * insert new record in front of lower priority
598                  */
599                 tp = sp->next;
600                 srp = sp->status_rec_list->status_record;
601                 sec = status_rec_p->uurec->sec;
602                 order = cmp_priority(severity, srp->severity, sec,
603                     srp->uurec->sec, np->max_pct, sp->max_pct);
604                 if (order > 0) {
605                         *rp = np;
606                 } else {
607                         srp = tp->status_rec_list->status_record;
608                         while (tp != sp &&
609                             cmp_priority(severity, srp->severity, sec,
610                             srp->uurec->sec, np->max_pct, sp->max_pct) < 0) {
611                                 tp = tp->next;
612                                 srp = tp->status_rec_list->status_record;
613                         }
614                 }
615                 np->next = tp;
616                 np->prev = tp->prev;
617                 tp->prev->next = np;
618                 tp->prev = np;
619         }
620 }

622 static void
623 add_resource_list(status_record_t *status_rec_p, name_list_t *fp,
624     resource_list_t **rpp)
625 {
626         int order;
627         resource_list_t *np, *end;
628         status_record_t *srp;

630         np = *rpp;
631         end = np;
632         while (np) {
633                 if (strcmp(fp->name, np->resource) == 0) {
634                         np->not_suppressed |= status_rec_p->not_suppressed;
635                         np->injected |= status_rec_p->injected;
636                         srp = np->status_rec_list->status_record;
637                         order = cmp_priority(status_rec_p->severity,
638                             srp->severity, status_rec_p->uurec->sec,
639                             srp->uurec->sec, fp->max_pct, np->max_pct);
640                         if (order > 0 && np != end) {
641                                 /*
642                                  * remove from list and add again using
643                                  * new priority
644                                  */
645                                 np->prev->next = np->next;
646                                 np->next->prev = np->prev;
647                                 add_resource(status_rec_p,
648                                     rpp, np);
649                         } else {
650                                 add_rec_list(status_rec_p,
651                                     &np->status_rec_list);
652                         }
653                         break;
654                 }
655                 np = np->next;
```

```
656                 if (np == end) {
657                         np = NULL;
658                         break;
659                 }
660         }
661         if (np == NULL) {
662                 np = malloc(sizeof (resource_list_t));
663                 np->resource = fp->name;
664                 np->not_suppressed = status_rec_p->not_suppressed;
665                 np->injected = status_rec_p->injected;
666                 np->status_rec_list = NULL;
667                 np->max_pct = fp->max_pct;
668                 add_resource(status_rec_p, rpp, np);
669         }
670 }

672 static void
673 add_list(status_record_t *status_rec_p, name_list_t *listp,
674     resource_list_t **glistp)
675 {
676         name_list_t *fp, *end;

678         fp = listp;
679         end = fp;
680         while (fp) {
681                 add_resource_list(status_rec_p, fp, glistp);
682                 fp = fp->next;
683                 if (fp == end)
684                         break;
685         }
686 }

688 /*
689  * add record to rec, fru and asru lists.
690  */
691 static void
692 catalog_new_record(uurec_t *uurec_p, char *msgid, name_list_t *class,
693     name_list_t *fru, name_list_t *asru, name_list_t *resource,
694     name_list_t *serial, boolean_t not_suppressed,
695     hostid_t *hostid, boolean_t injected, boolean_t dummy_fru)
696 {
697         status_record_t *status_rec_p;

699         status_rec_p = new_record_init(uurec_p, msgid, class, fru, asru,
700             resource, serial, not_suppressed, hostid, injected);
701         add_rec_list(status_rec_p, &status_rec_list);
702         if (status_rec_p->fru && !dummy_fru)
703                 add_list(status_rec_p, status_rec_p->fru, &status_fru_list);
704         if (status_rec_p->asru)
705                 add_list(status_rec_p, status_rec_p->asru, &status_asru_list);
706 }

708 static void
709 get_serial_no(nvlist_t *nvl, name_list_t **serial_p, uint8_t pct)
710 {
711         char *name;
712         char *serial = NULL;
713         char **lserial = NULL;
714         uint64_t serint;
715         name_list_t *nlp;
716         int j;
717         uint_t nelem;
718         char buf[64];

720         if (nvlist_lookup_string(nvl, FM_FMRI_SCHEME, &name) == 0) {
721                 if (strcmp(name, FM_FMRI_SCHEME_CPU) == 0) {
```



```
722                         if (nvlist_lookup_uint64(nvl, FM_FMRI_CPU_SERIAL_ID,
723                             &serint) == 0) {
724                                 (void) snprintf(buf, sizeof (buf), "%llX",
725                                     serint);
726                                 nlp = alloc_name_list(buf, pct);
727                                 (void) merge_name_list(serial_p, nlp, 1);
728                         }
729                 } else if (strcmp(name, FM_FMRI_SCHEME_MEM) == 0) {
730                         if (nvlist_lookup_string_array(nvl,
731                             FM_FMRI_MEM_SERIAL_ID, &lserial, &nelem) == 0) {
732                                 nlp = alloc_name_list(lserial[0], pct);
733                                 for (j = 1; j < nelem; j++) {
734                                         name_list_t *n1lp;
735                                         n1lp = alloc_name_list(lserial[j], pct);
736                                         (void) merge_name_list(&nlp, n1lp, 1);
737                                 }
738                                 (void) merge_name_list(serial_p, nlp, 1);
739                         }
740                 } else if (strcmp(name, FM_FMRI_SCHEME_HC) == 0) {
741                         if (nvlist_lookup_string(nvl, FM_FMRI_HC_SERIAL_ID,
742                             &serial) == 0) {
743                                 nlp = alloc_name_list(serial, pct);
744                                 (void) merge_name_list(serial_p, nlp, 1);
745                         }
746                 }
747         }
748 }

750 static void
751 extract_record_info(nvlist_t *nvl, name_list_t **class_p,
752     name_list_t **fru_p, name_list_t **serial_p, name_list_t **resource_p,
753     name_list_t **asru_p, boolean_t *dummy_fru, uint8_t status)
754 {
755         nvlist_t *lfru, *lasru, *rsrc;
756         name_list_t *nlp;
757         char *name;
758         uint8_t lpct = 0;
759         char *lclass = NULL;
760         char *label;

762         (void) nvlist_lookup_uint8(nvl, FM_FAULT_CERTAINTY, &lpct);
763         if (nvlist_lookup_string(nvl, FM_CLASS, &lclass) == 0) {
764                 nlp = alloc_name_list(lclass, lpct);
765                 (void) merge_name_list(class_p, nlp, 1);
766         }
767         if (nvlist_lookup_nvlist(nvl, FM_FAULT_FRU, &lfru) == 0) {
768                 name = get_nvl2str_topo(lfru);
769                 if (name != NULL) {
770                         nlp = alloc_name_list(name, lpct);
771                         nlp->status = status & ~(FM_SUSPECT_UNUSABLE |
772                             FM_SUSPECT_DEGRADED);
773                         free(name);
774                         if (nvlist_lookup_string(nvl, FM_FAULT_LOCATION,
775                             &label) == 0)
776                                 nlp->label = strdup(label);
777                         (void) merge_name_list(fru_p, nlp, 1);
778                 }
779                 get_serial_no(lfru, serial_p, lpct);
780         } else if (nvlist_lookup_nvlist(nvl, FM_FAULT_RESOURCE, &rsrc) != 0) {
781                 /*
782                  * No FRU or resource. But we want to display the repair status
783                  * somehow, so create a dummy FRU field.
784                  */
785                 *dummy_fru = 1;
786                 nlp = alloc_name_list(dgettext("FMD", "None"), lpct);
787                 nlp->status = status & ~(FM_SUSPECT_UNUSABLE |
```

```
788                     FM_SUSPECT_DEGRADED);
789                 (void) merge_name_list(fru_p, nlp, 1);
790         }
791         if (nvlist_lookup_nvlist(nvl, FM_FAULT_ASRU, &lasru) == 0) {
792                 name = get_nvl2str_topo(lasru);
793                 if (name != NULL) {
794                         nlp = alloc_name_list(name, lpct);
795                         nlp->status = status & ~(FM_SUSPECT_NOT_PRESENT |
796                             FM_SUSPECT_REPAIRED | FM_SUSPECT_REPLACED |
797                             FM_SUSPECT_ACQUITTED);
798                         free(name);
799                         (void) merge_name_list(asru_p, nlp, 1);
800                 }
801                 get_serial_no(lasru, serial_p, lpct);
802         }
803         if (nvlist_lookup_nvlist(nvl, FM_FAULT_RESOURCE, &rsrc) == 0) {
804                 name = get_nvl2str_topo(rsrc);
805                 if (name != NULL) {
806                         nlp = alloc_name_list(name, lpct);
807                         nlp->status = status;
808                         free(name);
809                         if (nvlist_lookup_string(nvl, FM_FAULT_LOCATION,
810                             &label) == 0)
811                                 nlp->label = strdup(label);
812                         (void) merge_name_list(resource_p, nlp, 1);
813                 }
814         }
815 }

816 static void
817 add_fault_record_to_catalog(nvlist_t *nvl, uint64_t sec, char *uuid)
818 {
819         char *msgid = "-";
820         uint_t i, size = 0;
821         name_list_t *class = NULL, *resource = NULL;
822         name_list_t *asru = NULL, *fru = NULL, *serial = NULL;
823         nvlist_t **nva;
824         uint8_t *ba;
825         uurec_t *uurec_p;
826         hostid_t *host;
827         boolean_t not_suppressed = 1;
828         boolean_t any_present = 0;
829         boolean_t injected = 0;
830         boolean_t dummy_fru = 0;

831         (void) nvlist_lookup_string(nvl, FM_SUSPECT_DIAG_CODE, &msgid);
832         (void) nvlist_lookup_uint32(nvl, FM_SUSPECT_FAULT_SZ, &size);
833         (void) nvlist_lookup_boolean_value(nvl, FM_SUSPECT_MESSAGE,
834             &not_suppressed);
835         (void) nvlist_lookup_boolean_value(nvl, FM_SUSPECT_INJECTED, &injected);

836         if (size != 0) {
837                 (void) nvlist_lookup_nvlist_array(nvl, FM_SUSPECT_FAULT_LIST,
838                     &nva, &size);
839                 (void) nvlist_lookup_uint8_array(nvl, FM_SUSPECT_FAULT_STATUS,
840                     &ba, &size);
841                 for (i = 0; i < size; i++) {
842                         extract_record_info(nva[i], &class, &fru, &serial,
843                             &resource, &asru, &dummy_fru, ba[i]);
844                         if (!(ba[i] & FM_SUSPECT_NOT_PRESENT) &&
845                             (ba[i] & FM_SUSPECT_FAULTY))
846                                 any_present = 1;
847                 }
848                 /*
849                  * also suppress if no resources present
850                  */
851                 if (any_present == 0)
852                         not_suppressed = 0;
853         }
```



```
854         uurec_p = (uurec_t *)malloc(sizeof (uurec_t));
855         uurec_p->uuid = strdup(uuid);
856         uurec_p->sec = sec;
857         uurec_p->ari_uuid_list = NULL;
858         uurec_p->event = NULL;
859         (void) nvlist_dup(nvl, &uurec_p->event, 0);
860         host = find_hostid(nvl);
861         catalog_new_record(uurec_p, msgid, class, fru, asru,
862             resource, serial, not_suppressed, host, injected, dummy_fru);
863 }

864 static void
865 update_asru_state_in_catalog(const char *uuid, const char *ari_uuid)
866 {
867         sr_list_t *srp;
868         uurec_t *uurp;
869         ari_list_t *ari_list;

870         srp = status_rec_list;
871         if (srp) {
872                 for (;;) {
873                         uurp = srp->status_record->uurec;
874                         while (uurp) {
875                                 if (strcmp(uuid, uurp->uuid) == 0) {
876                                         ari_list = (ari_list_t *)
877                                             malloc(sizeof (ari_list_t));
878                                         ari_list->ari_uuid = strdup(ari_uuid);
879                                         ari_list->next = uurp->ari_uuid_list;
880                                         uurp->ari_uuid_list = ari_list;
881                                         return;
882                                 }
883                                 uurp = uurp->next;
884                         }
885                         if (srp->next == status_rec_list)
886                                 break;
887                         srp = srp->next;
888                 }
889         }
890 }

891 static void
892 print_line(char *label, char *buf)
893 {
894         char *cp, *ep, *wp;
895         char c;
896         int i;
897         int lsz;
898         char *padding;

899         lsz = strlen(label);
900         padding = malloc(lsz + 1);
901         for (i = 0; i < lsz; i++)
902                 padding[i] = ' ';
903         padding[i] = 0;
904         cp = buf;
905         ep = buf;
906         c = *ep;
907         (void) printf("\n");
908         while (c) {
909                 i = lsz;
910                 wp = NULL;
911                 while ((c = *ep) != NULL && (wp == NULL || i < 80)) {
```

```
920                         if (c == ' ')
921                                 wp = ep;
922                         else if (c == '\n') {
923                                 i = 0;
924                                 *ep = 0;
925                                 do {
926                                         ep++;
927                                 } while ((c = *ep) != NULL && c == ' ');
928                                 break;
929                         }
930                         ep++;
931                         i++;
932                 }
933                 if (i >= 80 && wp) {
934                         *wp = 0;
935                         ep = wp + 1;
936                         c = *ep;
937                 }
938                 (void) printf("%s%s\n", label, cp);
939                 cp = ep;
940                 label = padding;
941         }
942         free(padding);
943 }

945 static void
946 print_dict_info_line(nvlist_t *e, fmd_msg_item_t what, const char *linehdr)
947 {
948         char *cp = fmd_msg_getitem_nv(fmadm_msghdl, NULL, e, what);

950         if (cp) {
951                 print_line(dgettext("FMD", linehdr), cp);
952                 free(cp);
953         }
954 }

956 static void
957 print_dict_info(nvlist_t *nvl)
958 {
959         print_dict_info_line(nvl, FMD_MSG_ITEM_DESC, "Description : ");
960         print_dict_info_line(nvl, FMD_MSG_ITEM_RESPONSE, "Response    : ");
961         print_dict_info_line(nvl, FMD_MSG_ITEM_IMPACT, "Impact      : ");
962         print_dict_info_line(nvl, FMD_MSG_ITEM_ACTION, "Action      : ");
963 }

965 static void
966 print_name(name_list_t *list, char *padding, int *np, int pct, int full)
967 {
968         char *name;

970         name = list->name;
971         if (list->label) {
972                 (void) printf("%s \"%s\" (%s)", padding, list->label, name);
973                 *np += 1;
974         } else {
975                 (void) printf("%s %s", padding, name);
976                 *np += 1;
977         }
978         if (list->pct && pct > 0 && pct < 100) {
979                 if (list->count > 1) {
980                         if (full) {
981                                 (void) printf(" %d @ %s %d%%\n", list->count,
982                                     dgettext("FMD", "max"),
983                                     list->max_pct);
984                         } else {
985                                 (void) printf(" %s %d%%\n",
```



```
986                                     dgettext("FMD", "max"),
987                                     list->max_pct);
988                         }
989                 } else {
990                         (void) printf(" %d%%\n", list->pct);
991                 }
992         } else {
993                 (void) printf("\n");
994         }
995 }

997 static void
998 print_asru_status(int status, char *label)
999 {
1000         char *msg = NULL;

1002         switch (status) {
1003         case 0:
1004                 msg = dgettext("FMD", "ok and in service");
1005                 break;
1006         case FM_SUSPECT_DEGRADED:
1007                 msg = dgettext("FMD", "service degraded, "
1008                     "but associated components no longer faulty");
1009                 break;
1010         case FM_SUSPECT_FAULTY | FM_SUSPECT_DEGRADED:
1011                 msg = dgettext("FMD", "faulted but still "
1012                     "providing degraded service");
1013                 break;
1014         case FM_SUSPECT_FAULTY:
1015                 msg = dgettext("FMD", "faulted but still in service");
1016                 break;
1017         case FM_SUSPECT_UNUSABLE:
1018                 msg = dgettext("FMD", "out of service, "
1019                     "but associated components no longer faulty");
1020                 break;
1021         case FM_SUSPECT_FAULTY | FM_SUSPECT_UNUSABLE:
1022                 msg = dgettext("FMD", "faulted and taken out of service");
1023                 break;
1024         default:
1025                 break;
1026         }
1027         if (msg) {
1028                 (void) printf("%s     %s\n", label, msg);
1029         }
1030 }

1032 static void
1033 print_fru_status(int status, char *label)
1034 {
1035         char *msg = NULL;

1037         if (status & FM_SUSPECT_NOT_PRESENT)
1038                 msg = dgettext("FMD", "not present");
1039         else if (status & FM_SUSPECT_FAULTY)
1040                 msg = dgettext("FMD", "faulty");
1041         else if (status & FM_SUSPECT_REPLACED)
1042                 msg = dgettext("FMD", "replaced");
1043         else if (status & FM_SUSPECT_REPAIRED)
1044                 msg = dgettext("FMD", "repair attempted");
1045         else if (status & FM_SUSPECT_ACQUITTED)
1046                 msg = dgettext("FMD", "acquitted");
1047         else
1048                 msg = dgettext("FMD", "removed");
1049         (void) printf("%s     %s\n", label, msg);
1050 }
```

```
1052 static void
1053 print_rsrc_status(int status, char *label)
1054 {
1055         char *msg = "";

1057         if (status & FM_SUSPECT_NOT_PRESENT)
1058                 msg = dgettext("FMD", "not present");
1059         else if (status & FM_SUSPECT_FAULTY) {
1060                 if (status & FM_SUSPECT_DEGRADED)
1061                         msg = dgettext("FMD",
1062                             "faulted but still providing degraded service");
1063                 else if (status & FM_SUSPECT_UNUSABLE)
1064                         msg = dgettext("FMD",
1065                             "faulted and taken out of service");
1066                 else
1067                         msg = dgettext("FMD", "faulted but still in service");
1068         } else if (status & FM_SUSPECT_REPLACED)
1069                 msg = dgettext("FMD", "replaced");
1070         else if (status & FM_SUSPECT_REPAIRED)
1071                 msg = dgettext("FMD", "repair attempted");
1072         else if (status & FM_SUSPECT_ACQUITTED)
1073                 msg = dgettext("FMD", "acquitted");
1074         else
1075                 msg = dgettext("FMD", "removed");
1076         (void) printf("%s     %s\n", label, msg);
1077 }

1079 static void
1080 print_name_list(name_list_t *list, char *label,
1081     int limit, int pct, void (func1)(int, char *), int full)
1082 {
1083         char *name;
1084         char *padding;
1085         int i, j, l, n;
1086         name_list_t *end = list;

1088         l = strlen(label);
1089         padding = malloc(l + 1);
1090         for (i = 0; i < l; i++)
1091                 padding[i] = ' ';
1092         padding[l] = 0;
1093         (void) printf("%s", label);
1094         name = list->name;
1095         if (list->label)
1096                 (void) printf(" \"%s\" (%s)", list->label, name);
1097         else
1098                 (void) printf(" %s", name);
1099         if (list->pct && pct > 0 && pct < 100) {
1100                 if (list->count > 1) {
1101                         if (full) {
1102                                 (void) printf(" %d @ %s %d%%\n", list->count,
1103                                     dgettext("FMD", "max"), list->max_pct);
1104                         } else {
1105                                 (void) printf(" %s %d%%\n",
1106                                     dgettext("FMD", "max"), list->max_pct);
1107                         }
1108                 } else {
1109                         (void) printf(" %d%%\n", list->pct);
1110                 }
1111         } else {
1112                 (void) printf("\n");
1113         }
1114         if (func1)
1115                 func1(list->status, padding);
1116         n = 1;
1117         j = 0;
```



```
1118         while ((list = list->next) != end) {
1119                 if (limit == 0 || n < limit) {
1120                         print_name(list, padding, &n, pct, full);
1121                         if (func1)
1122                                 func1(list->status, padding);
1123                 } else
1124                         j++;
1125         }
1126         if (j == 1) {
1127                 print_name(list->prev, padding, &n, pct, full);
1128         } else if (j > 1) {
1129                 (void) printf("%s... %d %s\n", padding, j,
1130                     dgettext("FMD", "more entries suppressed,"
1131                     " use -v option for full list"));
1132         }
1133         free(padding);
1134 }

1136 static int
1137 asru_same_status(name_list_t *list)
1138 {
1139         name_list_t *end = list;
1140         int status = list->status;

1142         while ((list = list->next) != end) {
1143                 if (status == -1) {
1144                         status = list->status;
1145                         continue;
1146                 }
1147                 if (list->status != -1 && status != list->status) {
1148                         status = -1;
1149                         break;
1150                 }
1151         }
1152         return (status);
1153 }

1155 static int
1156 serial_in_fru(name_list_t *fru, name_list_t *serial)
1157 {
1158         name_list_t *sp = serial;
1159         name_list_t *fp;
1160         int nserial = 0;
1161         int found = 0;
1162         char buf[128];

1164         while (sp) {
1165                 fp = fru;
1166                 nserial++;
1167                 (void) snprintf(buf, sizeof (buf), "serial=%s", sp->name);
1168                 buf[sizeof (buf) - 1] = 0;
1169                 while (fp) {
1170                         if (strstr(fp->name, buf) != NULL) {
1171                                 found++;
1172                                 break;
1173                         }
1174                         fp = fp->next;
1175                         if (fp == fru)
1176                                 break;
1177                 }
1178                 sp = sp->next;
1179                 if (sp == serial)
1180                         break;
1181         }
1182         return (found == nserial ? 1 : 0);
1183 }
```

```
1185 static void
1186 print_sup_record(status_record_t *srp, int opt_i, int full)
1187 {
1188         char buf[32];
1189         uurec_t *uurp = srp->uurec;
1190         int n, j, k, max;
1191         int status;
1192         ari_list_t *ari_list;

1194         n = 0;
1195         max = max_fault;
1196         if (max < 0) {
1197                 max = 0;
1198         }
1199         j = max / 2;
1200         max -= j;
1201         k = srp->nrecs - max;
1202         while ((uurp = uurp->next) != NULL) {
1203                 if (full || n < j || n >= k || max_fault == 0 ||
1204                     srp->nrecs == max_fault+1) {
1205                         if (opt_i) {
1206                                 ari_list = uurp->ari_uuid_list;
1207                                 while (ari_list) {
1208                                         (void) printf("%-15s %s\n",
1209                                             format_date(buf, sizeof (buf),
1210                                             uurp->sec), ari_list->ari_uuid);
1211                                         ari_list = ari_list->next;
1212                                 }
1213                         } else {
1214                                 (void) printf("%-15s %s\n",
1215                                     format_date(buf, sizeof (buf), uurp->sec),
1216                                     uurp->uuid);
1217                         }
1218                 } else if (n == j)
1219                         (void) printf("... %d %s\n", srp->nrecs - max_fault,
1220                             dgettext("FMD", "more entries suppressed"));
1221                 n++;
1222         }
1223         (void) printf("\n");
1224         (void) printf("%s %s", dgettext("FMD", "Host        :"),
1225             srp->host->server);
1226         if (srp->host->domain)
1227                 (void) printf("\t%s %s", dgettext("FMD", "Domain      :"),
1228                     srp->host->domain);
1229         (void) printf("\n%s %s", dgettext("FMD", "Platform    :"),
1230             srp->host->platform);
1231         (void) printf("\t%s %s", dgettext("FMD", "Chassis_id  :"),
1232             srp->host->chassis ? srp->host->chassis : "");
1233         (void) printf("\n%s %s\n\n", dgettext("FMD", "Product_sn  :"),
1234             srp->host->product_sn? srp->host->product_sn : "");
1235         if (srp->class)
1236                 print_name_list(srp->class,
1237                     dgettext("FMD", "Fault class :"), 0, srp->class->pct,
1238                     NULL, full);
1239         if (srp->asru) {
1240                 status = asru_same_status(srp->asru);
1241                 if (status != -1) {
1242                         print_name_list(srp->asru,
1243                             dgettext("FMD", "Affects     :"),
1244                             full ? 0 : max_display, 0, NULL, full);
1245                         print_asru_status(status, "             ");
1246                 } else
1247                         print_name_list(srp->asru,
1248                             dgettext("FMD", "Affects     :"),
1249                             full ? 0 : max_display, 0, print_asru_status, full);
```



```
1250         }
1251         if (full || srp->fru == NULL || srp->asru == NULL) {
1252                 if (srp->resource) {
1253                         status = asru_same_status(srp->resource);
1254                         if (status != -1) {
1255                                 print_name_list(srp->resource,
1256                                     dgettext("FMD", "Problem in  :"),
1257                                     full ? 0 : max_display, 0, NULL, full);
1258                                 print_rsrc_status(status, "             ");
1259                         } else
1260                                 print_name_list(srp->resource,
1261                                     dgettext("FMD", "Problem in  :"),
1262                                     full ? 0 : max_display, 0,
1263                                     print_rsrc_status, full);
1264                 }
1265         }
1266         if (srp->fru) {
1267                 status = asru_same_status(srp->fru);
1268                 if (status != -1) {
1269                         print_name_list(srp->fru, dgettext("FMD",
1270                             "FRU         :"), 0,
1271                             srp->fru->pct == 100 ? 100 : srp->fru->max_pct,
1272                             NULL, full);
1273                         print_fru_status(status, "             ");
1274                 } else
1275                         print_name_list(srp->fru, dgettext("FMD",
1276                             "FRU         :"), 0,
1277                             srp->fru->pct == 100 ? 100 : srp->fru->max_pct,
1278                             print_fru_status, full);
1279         }
1280         if (srp->serial && !serial_in_fru(srp->fru, srp->serial) &&
1281             !serial_in_fru(srp->asru, srp->serial)) {
1282                 print_name_list(srp->serial, dgettext("FMD", "Serial ID.  :"),
1283                     0, 0, NULL, full);
1284         }
1285         print_dict_info(srp->uurec->event);
1286         (void) printf("\n");
1287 }

1289 static void
1290 print_status_record(status_record_t *srp, int summary, int opt_i, int full)
1291 {
1292         char buf[32];
1293         uurec_t *uurp = srp->uurec;
1294         static int header = 0;
1295         char *head;
1296         ari_list_t *ari_list;

1298         if (!summary || !header) {
1299                 if (opt_i) {
1300                         head = "-------------- "
1301                             "------------------------------------  "
1302                             "-------------- ---------\n"
1303                             "TIME           CACHE-ID"
1304                             "                                  MSG-ID"
1305                             "\n         SEVERITY\n-------------- "
1306                             "------------------------------------ "
1307                             " -------------- ---------";
1308                 } else {
1309                         head = "-------------- "
1310                             "------------------------------------  "
1311                             "-------------- ---------\n"
1312                             "TIME           EVENT-ID"
1313                             "                                  MSG-ID"
1314                             "\n         SEVERITY\n-------------- "
1315                             "------------------------------------ "
```

```
1316                            " -------------- ---------";
1317                }
1318                (void) printf("%s\n", dgettext("FMD", head));
1319                header = 1;
1320        }
1321        if (opt_i) {
1322                ari_list = uurp->ari_uuid_list;
1323                while (ari_list) {
1324                        (void) printf("%-15s %-37s %-14s %-9s %s\n",
1325                            format_date(buf, sizeof (buf), uurp->sec),
1326                            ari_list->ari_uuid, srp->msgid, srp->severity,
1327                            srp->injected ? dgettext("FMD", "injected") : "");
1328                        ari_list = ari_list->next;
1329                }
1330        } else {
1331                (void) printf("%-15s %-37s %-14s %-9s %s\n",
1332                    format_date(buf, sizeof (buf), uurp->sec),
1333                    uurp->uuid, srp->msgid, srp->severity,
1334                    srp->injected ? dgettext("FMD", "injected") : "");
1335        }

1337        if (!summary)
1338                print_sup_record(srp, opt_i, full);
1339 }

1341 static void
1342 print_catalog(int summary, int opt_a, int full, int opt_i, int page_feed)
1343 {
1344        status_record_t *srp;
1345        sr_list_t *slp;

1347        slp = status_rec_list;
1348        if (slp) {
1349                for (;;) {
1350                        srp = slp->status_record;
1351                        if (opt_a || srp->not_suppressed) {
1352                                if (page_feed)
1353                                        (void) printf("\f\n");
1354                                print_status_record(srp, summary, opt_i, full);
1355                        }
1356                        if (slp->next == status_rec_list)
1357                                break;
1358                        slp = slp->next;
1359                }
1360        }
1361 }

1363 static name_list_t *
1364 find_fru(status_record_t *srp, char *resource)
1365 {
1366        name_list_t *rt = NULL;
1367        name_list_t *fru = srp->fru;

1369        while (fru) {
1370                if (strcmp(resource, fru->name) == 0) {
1371                        rt = fru;
1372                        break;
1373                }
1374                fru = fru->next;
1375                if (fru == srp->fru)
1376                        break;
1377        }
1378        return (rt);
1379 }

1381 static void
```



```
1382 print_fru_line(name_list_t *fru, char *uuid)
1383 {
1384        if (fru->pct == 100) {
1385                (void) printf("%s %d %s %d%%\n", uuid, fru->count,
1386                    dgettext("FMD", "suspects in this FRU total certainty"),
1387                    100);
1388        } else {
1389                (void) printf("%s %d %s %d%%\n", uuid, fru->count,
1390                    dgettext("FMD", "suspects in this FRU max certainty"),
1391                    fru->max_pct);
1392        }
1393 }

1395 static void
1396 print_fru(int summary, int opt_a, int opt_i, int page_feed)
1397 {
1398        resource_list_t *tp = status_fru_list;
1399        status_record_t *srp;
1400        sr_list_t *slp, *end;
1401        uurec_t *uurp;
1402        name_list_t *fru;
1403        int status;
1404        ari_list_t *ari_list;

1406        while (tp) {
1407                if (opt_a || tp->not_suppressed) {
1408                        if (page_feed)
1409                                (void) printf("\f\n");
1410                        if (!summary)
1411                                (void) printf("----------------------------"
1412                                    "-------------------------------------"
1413                                    "----------\n");
1414                        slp = tp->status_rec_list;
1415                        end = slp;
1416                        do {
1417                                srp = slp->status_record;
1418                                if (!srp->not_suppressed) {
1419                                        slp = slp->next;
1420                                        continue;
1421                                }
1422                                fru = find_fru(srp, tp->resource);
1423                                if (fru) {
1424                                        if (fru->label)
1425                                                (void) printf("\"%s\" (%s) ",
1426                                                    fru->label, fru->name);
1427                                        else
1428                                                (void) printf("%s ",
1429                                                    fru->name);
1430                                        break;
1431                                }
1432                                slp = slp->next;
1433                        } while (slp != end);

1435                        slp = tp->status_rec_list;
1436                        end = slp;
1437                        status = 0;
1438                        do {
1439                                srp = slp->status_record;
1440                                if (!srp->not_suppressed) {
1441                                        slp = slp->next;
1442                                        continue;
1443                                }
1444                                fru = srp->fru;
1445                                while (fru) {
1446                                        if (strcmp(tp->resource,
1447                                            fru->name) == 0)
```

```
1448                                        status |= fru->status;
1449                                fru = fru->next;
1450                                if (fru == srp->fru)
1451                                        break;
1452                        }
1453                        slp = slp->next;
1454                } while (slp != end);
1455                if (status & FM_SUSPECT_NOT_PRESENT)
1456                        (void) printf(dgettext("FMD", "not present"));
1457                else if (status & FM_SUSPECT_FAULTY)
1458                        (void) printf(dgettext("FMD", "faulty"));
1459                else if (status & FM_SUSPECT_REPLACED)
1460                        (void) printf(dgettext("FMD", "replaced"));
1461                else if (status & FM_SUSPECT_REPAIRED)
1462                        (void) printf(dgettext("FMD",
1463                            "repair attempted"));
1464                else if (status & FM_SUSPECT_ACQUITTED)
1465                        (void) printf(dgettext("FMD", "acquitted"));
1466                else
1467                        (void) printf(dgettext("FMD", "removed"));

1469                if (tp->injected)
1470                        (void) printf(dgettext("FMD", " injected\n"));
1471                else
1472                        (void) printf(dgettext("FMD", "\n"));

1474                slp = tp->status_rec_list;
1475                end = slp;
1476                do {
1477                        srp = slp->status_record;
1478                        if (!srp->not_suppressed) {
1479                                slp = slp->next;
1480                                continue;
1481                        }
1482                        uurp = srp->uurec;
1483                        fru = find_fru(srp, tp->resource);
1484                        if (fru) {
1485                                if (opt_i) {
1486                                        ari_list = uurp->ari_uuid_list;
1487                                        while (ari_list) {
1488                                                print_fru_line(fru,
1489                                                    ari_list->ari_uuid);
1490                                                ari_list =
1491                                                    ari_list->next;
1492                                        }
1493                                } else {
1494                                        print_fru_line(fru, uurp->uuid);
1495                                }
1496                        }
1497                        slp = slp->next;
1498                } while (slp != end);
1499                if (!summary) {
1500                        slp = tp->status_rec_list;
1501                        end = slp;
1502                        do {
1503                                srp = slp->status_record;
1504                                if (!srp->not_suppressed) {
1505                                        slp = slp->next;
1506                                        continue;
1507                                }
1508                                if (srp->serial &&
1509                                    !serial_in_fru(srp->fru,
1510                                    srp->serial)) {
1511                                        print_name_list(srp->serial,
1512                                            dgettext("FMD",
1513                                            "Serial ID.  :"),
```



```
1514                                                0, 0, NULL, 1);
1515                                        break;
1516                                }
1517                                slp = slp->next;
1518                        } while (slp != end);
1519                }
1520        }
1521        tp = tp->next;
1522        if (tp == status_fru_list)
1523                break;
1524    }
1525 }

1527 static void
1528 print_asru(int opt_a)
1529 {
1530        resource_list_t *tp = status_asru_list;
1531        status_record_t *srp;
1532        sr_list_t *slp, *end;
1533        char *msg;
1534        int status;
1535        name_list_t *asru;

1537        while (tp) {
1538                if (opt_a || tp->not_suppressed) {
1539                        status = 0;
1540                        slp = tp->status_rec_list;
1541                        end = slp;
1542                        do {
1543                                srp = slp->status_record;
1544                                if (!srp->not_suppressed) {
1545                                        slp = slp->next;
1546                                        continue;
1547                                }
1548                                asru = srp->asru;
1549                                while (asru) {
1550                                        if (strcmp(tp->resource,
1551                                            asru->name) == 0)
1552                                                status |= asru->status;
1553                                        asru = asru->next;
1554                                        if (asru == srp->asru)
1555                                                break;
1556                                }
1557                                slp = slp->next;
1558                        } while (slp != end);
1559                        switch (status) {
1560                        case 0:
1561                                msg = dgettext("FMD", "ok");
1562                                break;
1563                        case FM_SUSPECT_DEGRADED:
1564                                msg = dgettext("FMD", "degraded");
1565                                break;
1566                        case FM_SUSPECT_FAULTY | FM_SUSPECT_DEGRADED:
1567                                msg = dgettext("FMD", "degraded");
1568                                break;
1569                        case FM_SUSPECT_FAULTY:
1570                                msg = dgettext("FMD", "degraded");
1571                                break;
1572                        case FM_SUSPECT_UNUSABLE:
1573                                msg = dgettext("FMD", "unknown");
1574                                break;
1575                        case FM_SUSPECT_FAULTY | FM_SUSPECT_UNUSABLE:
1576                                msg = dgettext("FMD", "faulted");
1577                                break;
1578                        default:
1579                                msg = "";
```

```
1580                                    break;
1581                            }
1582                            (void) printf("%-69s %s", tp->resource, msg);
1583                            if (tp->injected)
1584                                    (void) printf(dgettext("FMD", " injected\n"));
1585                            else
1586                                    (void) printf(dgettext("FMD", "\n"));
1587                    }
1588                    tp = tp->next;
1589                    if (tp == status_asru_list)
1590                            break;
1591            }
1592    }

1594    static int
1595    uuid_in_list(char *uuid, uurec_select_t *uurecp)
1596    {
1597            while (uurecp) {
1598                    if (strcmp(uuid, uurecp->uuid) == 0)
1599                            return (1);
1600                    uurecp = uurecp->next;
1601            }
1602            return (0);
1603    }

1605    static int
1606    dfault_rec(const fmd_adm_caseinfo_t *acp, void *arg)
1607    {
1608            int64_t *diag_time;
1609            uint_t nelem;
1610            int rt = 0;
1611            char *uuid = "-";
1612            uurec_select_t *uurecp = (uurec_select_t *)arg;

1614            if (nvlist_lookup_int64_array(acp->aci_event, FM_SUSPECT_DIAG_TIME,
1615                &diag_time, &nelem) == 0 && nelem >= 2) {
1616                    (void) nvlist_lookup_string(acp->aci_event, FM_SUSPECT_UUID,
1617                        &uuid);
1618                    if (uurecp == NULL || uuid_in_list(uuid, uurecp))
1619                            add_fault_record_to_catalog(acp->aci_event, *diag_time,
1620                                uuid);
1621            } else {
1622                    rt = -1;
1623            }
1624            return (rt);
1625    }

1627    /*ARGSUSED*/
1628    static int
1629    dstatus_rec(const fmd_adm_rsrcinfo_t *ari, void *unused)
1630    {
1631            update_asru_state_in_catalog(ari->ari_case, ari->ari_uuid);
1632            return (0);
1633    }

1635    static int
1636    get_cases_from_fmd(fmd_adm_t *adm, uurec_select_t *uurecp, int opt_i)
1637    {
1638            int rt = FMADM_EXIT_SUCCESS;

1640            /*
1641             * These calls may fail with Protocol error if message payload is
1642             * too big
1643             */
1644            if (fmd_adm_case_iter(adm, NULL, dfault_rec, uurecp) != 0)
1645                    die("failed to get case list from fmd");
```



```
1646            if (opt_i && fmd_adm_rsrc_iter(adm, 1, dstatus_rec, NULL) != 0)
1647                    die("failed to get case status from fmd");
1648            return (rt);
1649    }

1651    /*
1652     * fmadm faulty command
1653     *
1654     *      -a              show hidden fault records
1655     *      -f              show faulty fru's
1656     *      -g              force grouping of similar faults on the same fru
1657     *      -n              number of fault records to display
1658     *      -p              pipe output through pager
1659     *      -r              show faulty asru's
1660     *      -s              print summary of first fault
1661     *      -u              print listed uuid's only
1662     *      -v              full output
1663     */

1665    int
1666    cmd_faulty(fmd_adm_t *adm, int argc, char *argv[])
1667    {
1668            int opt_a = 0, opt_v = 0, opt_p = 0, opt_s = 0, opt_r = 0, opt_f = 0;
1669            int opt_i = 0;
1670            char *pager;
1671            FILE *fp;
1672            int rt, c, stat;
1673            uurec_select_t *tp;
1674            uurec_select_t *uurecp = NULL;

1676            while ((c = getopt(argc, argv, "afgin:prsu:v")) != EOF) {
1677                    switch (c) {
1678                    case 'a':
1679                            opt_a++;
1680                            break;
1681                    case 'f':
1682                            opt_f++;
1683                            break;
1684                    case 'g':
1685                            opt_g++;
1686                            break;
1687                    case 'i':
1688                            opt_i++;
1689                            break;
1690                    case 'n':
1691                            max_fault = atoi(optarg);
1692                            break;
1693                    case 'p':
1694                            opt_p++;
1695                            break;
1696                    case 'r':
1697                            opt_r++;
1698                            break;
1699                    case 's':
1700                            opt_s++;
1701                            break;
1702                    case 'u':
1703                            tp = (uurec_select_t *)malloc(sizeof (uurec_select_t));
1704                            tp->uuid = optarg;
1705                            tp->next = uurecp;
1706                            uurecp = tp;
1707                            opt_a = 1;
1708                            break;
1709                    case 'v':
1710                            opt_v++;
1711                            break;
```

```
1712                 default:
1713                         return (FMADM_EXIT_USAGE);
1714                 }
1715         }
1716         if (optind < argc)
1717                 return (FMADM_EXIT_USAGE);

1719         if ((fmadm_msghdl = fmd_msg_init(NULL, FMD_MSG_VERSION)) == NULL)
1720                 return (FMADM_EXIT_ERROR);
1721         rt = get_cases_from_fmd(adm, uurecp, opt_i);
1722         if (opt_p) {
1723                 if ((pager = getenv("PAGER")) == NULL)
1724                         pager = "/usr/bin/more";
1725                 fp = popen(pager, "w");
1726                 if (fp == NULL) {
1727                         rt = FMADM_EXIT_ERROR;
1728                         opt_p = 0;
1729                 } else {
1730                         (void) dup2(fileno(fp), 1);
1731                         setbuf(stdout, NULL);
1732                         (void) fclose(fp);
1733                 }
1734         }
1735         max_display = max_fault;
1736         if (opt_f)
1737                 print_fru(opt_s, opt_a, opt_i, opt_p && !opt_s);
1738         if (opt_r)
1739                 print_asru(opt_a);
1740         if (opt_f == 0 && opt_r == 0)
1741                 print_catalog(opt_s, opt_a, opt_v, opt_i, opt_p && !opt_s);
1742         fmd_msg_fini(fmadm_msghdl);
1743         if (topo_handle)
1744                 topo_close(topo_handle);
1745         if (opt_p) {
1746                 (void) fclose(stdout);
1747                 (void) wait(&stat);
1748         }
1749         return (rt);
1750 }

1752 int
1753 cmd_flush(fmd_adm_t *adm, int argc, char *argv[])
1754 {
1755         int i, status = FMADM_EXIT_SUCCESS;

1757         if (argc < 2 || (i = getopt(argc, argv, "")) != EOF)
1758                 return (FMADM_EXIT_USAGE);

1760         for (i = 1; i < argc; i++) {
1761                 if (fmd_adm_rsrc_flush(adm, argv[i]) != 0) {
1762                         warn("failed to flush %s", argv[i]);
1763                         status = FMADM_EXIT_ERROR;
1764                 } else
1765                         note("flushed resource history for %s\n", argv[i]);
1766         }

1768         return (status);
1769 }

1771 int
1772 cmd_repair(fmd_adm_t *adm, int argc, char *argv[])
1773 {
1774         int err;

1776         if (getopt(argc, argv, "") != EOF)
1777                 return (FMADM_EXIT_USAGE);
```



```
1779         if (argc - optind != 1)
1780                 return (FMADM_EXIT_USAGE);

1782         /*
1783          * argument could be a uuid, an fmri (asru, fru or resource)
1784          * or a label. Try uuid first, If that fails try the others.
1785          */
1786         err = fmd_adm_case_repair(adm, argv[optind]);
1787         if (err != 0)
1788                 err = fmd_adm_rsrc_repaired(adm, argv[optind]);

1790         if (err != 0)
1791                 die("failed to record repair to %s", argv[optind]);

1793         note("recorded repair to %s\n", argv[optind]);
1794         return (FMADM_EXIT_SUCCESS);
1795 }

1797 int
1798 cmd_repaired(fmd_adm_t *adm, int argc, char *argv[])
1799 {
1800         int err;

1802         if (getopt(argc, argv, "") != EOF)
1803                 return (FMADM_EXIT_USAGE);

1805         if (argc - optind != 1)
1806                 return (FMADM_EXIT_USAGE);

1808         /*
1809          * argument could be an fmri (asru, fru or resource) or a label.
1810          */
1811         err = fmd_adm_rsrc_repaired(adm, argv[optind]);
1812         if (err != 0)
1813                 die("failed to record repair to %s", argv[optind]);

1815         note("recorded repair to of %s\n", argv[optind]);
1816         return (FMADM_EXIT_SUCCESS);
1817 }

1819 int
1820 cmd_replaced(fmd_adm_t *adm, int argc, char *argv[])
1821 {
1822         int err;

1824         if (getopt(argc, argv, "") != EOF)
1825                 return (FMADM_EXIT_USAGE);

1827         if (argc - optind != 1)
1828                 return (FMADM_EXIT_USAGE);

1830         /*
1831          * argument could be an fmri (asru, fru or resource) or a label.
1832          */
1833         err = fmd_adm_rsrc_replaced(adm, argv[optind]);
1834         if (err != 0)
1835                 die("failed to record replacement of %s", argv[optind]);

1837         note("recorded replacement of %s\n", argv[optind]);
1838         return (FMADM_EXIT_SUCCESS);
1839 }

1841 int
1842 cmd_acquit(fmd_adm_t *adm, int argc, char *argv[])
1843 {
```

```
1844         int err;
1846         if (getopt(argc, argv, "") != EOF)
1847                 return (FMADM_EXIT_USAGE);
1849         if (argc - optind != 1 && argc - optind != 2)
1850                 return (FMADM_EXIT_USAGE);
1852         /*
1853          * argument could be a uuid, an fmri (asru, fru or resource)
1854          * or a label. Or it could be a uuid and an fmri or label.
1855          */
1856         if (argc - optind == 2) {
1857                 err = fmd_adm_rsrc_acquit(adm, argv[optind], argv[optind + 1]);
1858                 if (err != 0)
1859                         err = fmd_adm_rsrc_acquit(adm, argv[optind + 1],
1860                             argv[optind]);
1861         } else {
1862                 err = fmd_adm_case_acquit(adm, argv[optind]);
1863                 if (err != 0)
1864                         err = fmd_adm_rsrc_acquit(adm, argv[optind], "");
1865         }
1867         if (err != 0)
1868                 die("failed to record acquital of %s", argv[optind]);
1870         note("recorded acquital of %s\n", argv[optind]);
1871         return (FMADM_EXIT_SUCCESS);
1872 }
```

****
12383 Thu Sep 5 23:02:03 2013
new/usr/src/cmd/mdb/common/modules/ii/ii.c
3373 gcc >= 4.5 concerns about offsetof()
****

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/types.h>
#include <sys/mdb_modapi.h>

#include <sys/nsctl/nsctl.h>
#include <sys/unistat/spcs_s.h>
#include <sys/unistat/spcs_s_k.h>


#include <sys/nsctl/dsw.h>
#include <sys/nsctl/dsw_dev.h>

#include <sys/nsctl/nsvers.h>

#if defined(__GNUC__)
#define offsetof(s, m)  __builtin_offsetof(s, m)
#else
#define offsetof(s, m)  ((size_t)(&(((s *)0)->m)))
#endif
#define offsetof(s, m)  ((size_t)(&((s *)0)->m))


const mdb_bitmask_t bi_flags_bits[] = {
        { "DSW_GOLDEN", DSW_GOLDEN, DSW_GOLDEN },
        { "DSW_COPYINGP", DSW_COPYINGP, DSW_COPYINGP },
        { "DSW_COPYINGM", DSW_COPYINGM, DSW_COPYINGM },
        { "DSW_COPYINGS", DSW_COPYINGS, DSW_COPYINGS },
        { "DSW_COPYINGX", DSW_COPYINGX, DSW_COPYINGX },
        { "DSW_BMPOFFLINE", DSW_BMPOFFLINE, DSW_BMPOFFLINE },
        { "DSW_SHDOFFLINE", DSW_SHDOFFLINE, DSW_SHDOFFLINE },
        { "DSW_MSTOFFLINE", DSW_MSTOFFLINE, DSW_MSTOFFLINE },
        { "DSW_OVROFFLINE", DSW_OVROFFLINE, DSW_OVROFFLINE },
        { "DSW_TREEMAP", DSW_TREEMAP, DSW_TREEMAP },
        { "DSW_OVERFLOW", DSW_OVERFLOW, DSW_OVERFLOW },
        { "DSW_SHDEXPORT", DSW_SHDEXPORT, DSW_SHDEXPORT },
        { "DSW_SHDIMPORT", DSW_SHDIMPORT, DSW_SHDIMPORT },
        { "DSW_VOVERFLOW", DSW_VOVERFLOW, DSW_VOVERFLOW },
```



```
        { "DSW_HANGING", DSW_HANGING, DSW_HANGING },
        { "DSW_CFGOFFLINE", DSW_CFGOFFLINE, DSW_CFGOFFLINE },
        { "DSW_OVRHDRDRTY", DSW_OVRHDRDRTY, DSW_OVRHDRDRTY },
        { "DSW_RESIZED", DSW_RESIZED, DSW_RESIZED },
        { "DSW_FRECLAIM", DSW_FRECLAIM, DSW_FRECLAIM },
        { NULL, 0, 0 }
};
```

*unchanged_portion_omitted_*

```
**********************************************************
    1783 Thu Sep  5 23:02:04 2013
new/usr/src/cmd/mdb/common/modules/libumem/misc.h
3373 gcc >= 4.5 concerns about offsetof()
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _MDBMOD_MISC_H
#define _MDBMOD_MISC_H

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

#include <mdb/mdb_modapi.h>

#ifdef __cplusplus
extern "C" {
#endif

#if defined(__GNUC__)
#define offsetof(s, m)  __builtin_offsetof(s, m)
#else
#endif /* ! codereview */
#define offsetof(s, m)  ((size_t)(&(((s *)0)->m)))
#endif
#endif /* ! codereview */

extern int umem_debug(uintptr_t, uint_t, int, const mdb_arg_t *);

extern int umem_set_standalone(void);
extern ssize_t umem_lookup_by_name(const char *, GElf_Sym *);
extern ssize_t umem_readvar(void *, const char *);

/*
 * Returns non-zero if sym matches libumem*'prefix*
 */
int is_umem_sym(const char *, const char *);

#define dprintf(x) if (umem_debug_level) { \
        mdb_printf("umem debug: ");  \
        /*CSTYLED*/\
        mdb_printf x ;\
}
```



```
#define dprintf_cont(x) if (umem_debug_level) { \
        /*CSTYLED*/\
        mdb_printf x ;\
}

extern int umem_debug_level;

#ifdef __cplusplus
}
#endif

#endif  /* _MDBMOD_MISC_H */
```

```
**********************************************************
   27290 Thu Sep  5 23:02:05 2013
new/usr/src/cmd/pools/poolstat/poolstat.c
3373 gcc >= 4.5 concerns about offsetof()
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * poolstat - report active pool statistics
 */
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <unistd.h>
#include <locale.h>
#include <string.h>
#include <ctype.h>
#include <limits.h>
#include <errno.h>

#include <pool.h>
#include "utils.h"
#include "poolstat.h"
#include "poolstat_utils.h"
#include "statcommon.h"

#ifndef TEXT_DOMAIN
#define TEXT_DOMAIN     "SYS_TEST"
#endif

/* calculate offset of a particular element in a structure      */
#if defined(__GNUC__)
#define offsetof(s, m)  __builtin_offsetof(s, m)
#else
#endif /* ! codereview */
#define offsetof(s, m)  ((size_t)(&(((s *)0)->m)))
#endif
#endif /* ! codereview */
#define addrof(s)  ((char **)&(s))

/* verify if a field is printable in respect of the current option flags */
#define PRINTABLE(i)    ((lf->plf_ffs[(i)].pff_prt & D_FIELD) || \
        (lf->plf_ffs[(i)].pff_prt & X_FIELD))
```



```
typedef int (* formatter) (char *, int, int, poolstat_field_format_t *, char *);

static uint_t timestamp_fmt = NODATE;

/* available field formatters   */
static int default_f(char *, int, int, poolstat_field_format_t *, char *);
static int bigno_f(char *, int, int, poolstat_field_format_t *, char *);
static int used_stat_f(char *, int, int, poolstat_field_format_t *, char *);
static int header_f(char *, int, int, poolstat_field_format_t *, char *);

/* statistics bags used to collect data from various provider   */
static statistic_bag_t  pool_sbag_s;
static statistic_bag_t  pset_sbag_s;
static statistic_bag_t  *pool_sbag = &pool_sbag_s;
static statistic_bag_t  *pset_sbag = &pset_sbag_s;

/* formatter objects for pset, defined in a default printing sequence   */
static poolstat_field_format_t pset_ffs[] = {
        /* prt flags,name,header,type,width,minwidth,offset,formatter   */
        { DX_FIELD, "id", "id", LL, 3, 1, addrof(pool_sbag),
                offsetof(statistic_bag_t, sb_sysid),
                (formatter)default_f },
        { DX_FIELD, "pool", "pool", STR, 20, 14, addrof(pool_sbag),
                offsetof(statistic_bag_t, sb_name),
                (formatter)default_f },
        { DX_FIELD, "type", "type", STR, 4, 5, addrof(pset_sbag),
                offsetof(statistic_bag_t, sb_type),
                (formatter)default_f },
        { D_FIELD, "rid", "rid", LL, 3, 1, addrof(pset_sbag_s.bag),
                offsetof(pset_statistic_bag_t, pset_sb_sysid),
                (formatter)default_f },
        { DX_FIELD, "rset", "rset", STR, 20, 14, addrof(pset_sbag),
                offsetof(statistic_bag_t, sb_name),
                (formatter)default_f },
        { DX_FIELD, "min", "min", ULL, 4, 1, addrof(pset_sbag_s.bag),
                offsetof(pset_statistic_bag_t, pset_sb_min),
                (formatter)bigno_f },
        { DX_FIELD, "max", "max", ULL, 4, 1, addrof(pset_sbag_s.bag),
                offsetof(pset_statistic_bag_t, pset_sb_max),
                (formatter)bigno_f },
        { DX_FIELD, "size", "size", ULL, 4, 1, addrof(pset_sbag_s.bag),
                offsetof(pset_statistic_bag_t, pset_sb_size),
                (formatter)default_f },
        { DX_FIELD, "used", "used", FL, 4, -1, addrof(pset_sbag_s.bag),
                offsetof(pset_statistic_bag_t, pset_sb_used),
                (formatter)used_stat_f },
        { DX_FIELD, "load", "load", FL, 4, -1, addrof(pset_sbag_s.bag),
                offsetof(pset_statistic_bag_t, pset_sb_load),
                (formatter)default_f }
};

/* formatter objects for pool, defined in a default printing sequence   */
static poolstat_field_format_t pool_ffs[] = {
        /* prt flags,name,header,type,width,minwidth,offset,formatter   */
        { D_FIELD, "id", "id", LL, 3, 1, addrof(pool_sbag),
                offsetof(statistic_bag_t, sb_sysid),
                (formatter)default_f },
        { D_FIELD, "pool", "pool", STR, 20, 13, addrof(pool_sbag),
                offsetof(statistic_bag_t, sb_name),
                (formatter)default_f },
        { D_FIELD, "p_size", "size", ULL, 4, 1, addrof(pset_sbag_s.bag),
                offsetof(pset_statistic_bag_t, pset_sb_size),
                (formatter)default_f },
        { D_FIELD, "p_used", "used", FL, 4, -1, addrof(pset_sbag_s.bag),
                offsetof(pset_statistic_bag_t, pset_sb_used),
```

```
128                 (formatter)default_f },
129         { D_FIELD, "p_load", "load", FL, 4, -1, addrof(pset_sbag_s.bag),
130                 offsetof(pset_statistic_bag_t, pset_sb_load),
131                 (formatter)default_f },
132 };

134 /* lists with formatter objects, one for each statistics field */
135 static poolstat_line_format_t   pool_lf; /* formatting list in default mode */
136 static poolstat_line_format_t   pset_lf; /* formatting list for psets    */

138 /* name of pools to be shown */
139 static poolstat_list_element_t  *pnames;
140 /*
141  * type of resources to be shown, currently we only have one type 'pset'
142  * but, poolstat can be extended to handle new upcoming resource types.
143  */
144 static poolstat_list_element_t   *rtypes;

146 /* a handle to the pool configuration   */
147 static pool_conf_t *conf;

149 /* option flags         */
150 static int      rflag;
151 static int      pflag;
152 static int      oflag;

154 /* operands     */
155 static int      interval = 0;   /* update interval      */
156 static long     count    = 1;   /* one run              */

158 /* data structure handlers      */
159 static poolstat_list_element_t *
160         create_prt_sequence_list(char *, poolstat_line_format_t *);
161 static poolstat_list_element_t *
162         create_args_list(char *, poolstat_list_element_t *, const char *);

164 /* statistics update function   */
165 static void sa_update(statistic_bag_t *, int);

167 /* statistics printing function */
168 static void prt_pool_stats(poolstat_list_element_t *);

170 static void
171 usage(void)
172 {
173         (void) fprintf(stderr, gettext(
174 "Usage:\n"
175 "poolstat [-p pool-list] [-r rset-list] [-T d|u] [interval [count]]\n"
176 "poolstat [-p pool-list] [-o format -r rset-list] [-T d|u] [interval [count]]\n"
177 "  \'pool-list\' is a space-separated list of pool IDs or names\n"
178 "  \'rset-list\' is \'all\' or \'pset\'\n"
179 "  \'format\' for all resource types is one or more of:\n"
180 "\tid pool type rid rset min max size used load\n"));
181         (void) exit(E_USAGE);
182 }

184 static int
185 Atoi(char *p, int *errp)
186 {
187         int i;
188         char *q;
189         errno = 0;
190         i = strtol(p, &q, 10);
191         if (errno != 0 || q == p || *q != '\0')
192                 *errp = -1;
193         else
```



```
194                 *errp = 0;
195         return (i);
196 }

198 int
199 main(int argc, char *argv[])
200 {
201         char            c;
202         int             error = 0;

204         (void) getpname(argv[0]);
205         (void) setlocale(LC_ALL, "");
206         (void) textdomain(TEXT_DOMAIN);

208         /* pset_sbag_s is used to collect pset statistics   */
209         pset_sbag_s.sb_type = PSET_TYPE_NAME;
210         pset_sbag_s.bag = ZALLOC(sizeof (pset_statistic_bag_t));
211         pool_sbag_s.sb_type = POOL_TYPE_NAME;

213         pset_lf.plf_ffs = pset_ffs;
214         pset_lf.plf_ff_len = sizeof (pset_ffs) /
215             sizeof (poolstat_field_format_t);
216         pool_lf.plf_ffs = pool_ffs;
217         pool_lf.plf_ff_len = sizeof (pool_ffs) /
218             sizeof (poolstat_field_format_t);

220         /* Don't let buffering interfere with piped output. */
221         (void) setvbuf(stdout, NULL, _IOLBF, 0);

223         while ((c = getopt(argc, argv, ":p:r:o:T:")) != EOF) {
224                 switch (c) {
225                 case 'p':       /* pool name specification      */
226                         pflag++;
227                         pnames = create_args_list(optarg, pnames,
228                             " \t");
229                         break;
230                 case 'r': {     /* resource type                */
231                         rflag++;
232                         rtypes = create_args_list(optarg, rtypes,
233                             " \t,");
234                         break;
235                         }
236                 case 'o': {     /* format specification         */
237                         oflag++;
238                         if (create_prt_sequence_list(optarg, &pset_lf) == NULL)
239                                 usage();
240                         break;
241                         }
242                 case 'T':
243                         if (optarg) {
244                                 if (*optarg == 'u')
245                                         timestamp_fmt = UDATE;
246                                 else if (*optarg == 'd')
247                                         timestamp_fmt = DDATE;
248                                 else
249                                         usage();
250                         } else {
251                                         usage();
252                         }
253                         break;
254                 case ':': {
255                         (void) fprintf(stderr,
256                             gettext(ERR_OPTION_ARGS), optopt);
257                         usage();
258                         /*NOTREACHED*/
259                         }
```

```
260                 default:
261                         (void) fprintf(stderr, gettext(ERR_OPTION), optopt);
262                         usage();
263                         /*NOTREACHED*/
264                 }
265         }

267         /* get operands */
268         if (argc > optind) {
269                 if ((interval = Atoi(argv[optind++], &error)) < 1 || error != 0)
270                         usage();
271                 count = -1;
272         }
273         if (argc > optind) {
274                 if ((count = Atoi(argv[optind++], &error)) < 1 || error != 0)
275                         usage();
276         }
277         /* check for extra options/operands     */
278         if (argc > optind)
279                 usage();

281         /* check options        */
282         if (oflag && !rflag)
283                 usage();

285         /* global initializations       */
286         if (!oflag) {
287                 /* create the default print sequences   */
288                 (void) create_prt_sequence_list(NULL, &pool_lf);
289                 (void) create_prt_sequence_list(NULL, &pset_lf);
290         }

292         if (rtypes == NULL || strcmp(rtypes->ple_obj, "all") == 0) {
293                 /* crate a default resource list        */
294                 FREE(rtypes);
295                 rtypes = create_args_list("pset", NULL, " \t,");
296         }

298         if ((conf = pool_conf_alloc()) == NULL)
299                 die(gettext(ERR_NOMEM));
300         if (pool_conf_open(conf, pool_dynamic_location(), PO_RDONLY)
301             != PO_SUCCESS)
302                 die(gettext(ERR_OPEN_DYNAMIC), get_errstr());

304         /* initialize statistic adapters        */
305         sa_libpool_init(conf);
306         sa_kstat_init(NULL);

308         /* collect and print out statistics     */
309         while (count-- != 0) {
310                 sa_update(pool_sbag, SA_REFRESH);
311                 if (timestamp_fmt != NODATE)
312                         print_timestamp(timestamp_fmt);
313                 if (pool_sbag->sb_changed & POU_POOL)
314                         (void) printf(
315                         "<<State change>>\n");
316                 prt_pool_stats(pnames);
317                 if (count != 0) {
318                         (void) sleep(interval);
319                         if (rflag)
320                                 (void) printf("\n");
321                 }
322         }

324         return (E_PO_SUCCESS);
325 }
```



```
327 /*
328  * Take the arguments and create/append a string list to the 'le' list.
329  */
330 static poolstat_list_element_t  *
331 create_args_list(char *arg, poolstat_list_element_t  *le, const char *delim)
332 {
333         poolstat_list_element_t *head = le;

335         while (arg != NULL && *arg != '\0') {
336                 char *name = arg;
337                 arg = strpbrk(arg, delim);
338                 if (arg != NULL) {
339                         *arg++ = '\0';
340                 }
341                 if (le == NULL) {
342                         /* create first element */
343                         NEW0(le);
344                         head = le;
345                 } else {
346                         /* find last and append */
347                         while (le->ple_next != NULL)
348                                 le = le->ple_next;
349                         NEW0(le->ple_next);
350                         le = le->ple_next;
351                 }
352                 le->ple_obj = (void *)name;
353         }

355         return (head);
356 }

358 /*
359  * Take the arguments to the -o option, and create a format field list in order
360  * specified by 'arg'.
361  * If 'arg' is NULL a list in a default printing order is created.
362  */
363 static poolstat_list_element_t *
364 create_prt_sequence_list(char *arg, poolstat_line_format_t *lf)
365 {
366         /*
367          * Create a default print sequence. It is the sequence defined
368          * statically in the format list. At the same time mark the fields
369          * printable according to the current option settings.
370          */
371         if (arg == NULL) {
372                 int     i;
373                 NEW0(lf->plf_prt_seq);
374                 lf->plf_ffs[0].pff_prt |= PRINTABLE(0) ? PABLE_FIELD : 0;
375                 lf->plf_last = lf->plf_prt_seq;
376                 lf->plf_last->ple_obj = &(lf->plf_ffs[0]);
377                 for (i = 1; i < lf->plf_ff_len; i++) {
378                         lf->plf_ffs[i].pff_prt |=
379                             PRINTABLE(i) ? PABLE_FIELD : 0;
380                         NEW0(lf->plf_last->ple_next);
381                         lf->plf_last = lf->plf_last->ple_next;
382                         lf->plf_last->ple_obj = &(lf->plf_ffs[i]);
383                 }
384                 return (lf->plf_prt_seq);
385         }

387         while (arg != NULL && *arg != '\0') {
388                 poolstat_field_format_t *ff;    /* current format field */
389                 int     ffIdx;  /* format field index       */
390                 char    *name;  /* name of field            */
391                 int     n;      /* no. of chars to strip    */
```

```
393                 n = strspn(arg, " ,\t\r\v\f\n");
394                 arg += n;       /* strip multiples separator    */
395                 name = arg;

397                 if (strlen(name) < 1)
398                         break;

400                 if ((arg = strpbrk(arg, " ,\t\r\v\f\n")) != NULL)
401                         *arg++ = '\0';

403                 /* search for a named format field */
404                 for (ffIdx = 0; ffIdx < lf->plf_ff_len; ffIdx++) {
405                         ff = lf->plf_ffs + ffIdx;
406                         if (strcmp(ff->pff_name, name) == 0) {
407                                 ff->pff_prt |= PABLE_FIELD;
408                                 break;
409                         }
410                 }
411                 /* if the name wasn't found     */
412                 if (ffIdx == lf->plf_ff_len) {
413                         (void) fprintf(stderr, gettext(ERR_UNSUPP_STAT_FIELD),
414                             name);
415                         usage();
416                 }
417                 if (lf->plf_last == NULL) {
418                         /* create first print handle */
419                         NEW0(lf->plf_prt_seq);
420                         lf->plf_last = lf->plf_prt_seq;
421                 } else {
422                         NEW0(lf->plf_last->ple_next);
423                         lf->plf_last = lf->plf_last->ple_next;
424                 }
425                 lf->plf_last->ple_obj = ff;     /* refer to the format field */
426         }

428         return (lf->plf_prt_seq);
429 }

431 /* update the statistic data by adapters        */
432 static void
433 sa_update(statistic_bag_t *sbag, int flags)
434 {
435         sa_libpool_update(sbag, flags);
436         sa_kstat_update(sbag, flags);
437 }

439 /*
440  * Format one statistic field and put it into the 'str' buffer. 'ff' contains
441  * the field formatting parameters. Return the number of used bytes.
442  */
443 static int
444 default_f(char *str, int pos, int left, poolstat_field_format_t *ff, char *data)
445 {
446         int  used;

448         switch (ff->pff_type) {
449         case LL: {
450                         int64_t v;
451                         v = *((int64_t *)(void *)(data + ff->pff_offset));
452                         used = snprintf(str + pos, left, "%*.*lld",
453                             ff->pff_width, ff->pff_minwidth, v);
454                 }
455                 break;
456         case ULL: {
457                         uint64_t v;
```



```
458                         v = *((uint64_t *)(void *)(data + ff->pff_offset));
459                         used = snprintf(str + pos, left, "%*.*llu",
460                             ff->pff_width, ff->pff_minwidth, v);
461                 };
462                 break;
463         case FL: {
464                         int     pw = 0;
465                         double v = *((double *)(void *)(data + ff->pff_offset));
466                         if (v < 10) {
467                                 pw = ff->pff_width - 2;
468                         } else if (v < 100) {
469                                 pw = ff->pff_width - 3;
470                         } else if (v < 1000) {
471                                 pw = ff->pff_width - 4;
472                         }
473                         if (pw < 0)
474                                 pw = 0;
475                         used = snprintf(str + pos, left, "%*.*f",
476                             ff->pff_width, pw, v);
477                 };
478                 break;
479         case STR: {
480                         char    *v;
481                         int     sl;
482                         v = *((char **)(void *)(data + ff->pff_offset));
483                         sl = strlen(v);
484                         /* truncate if it doesn't fit   */
485                         if (sl > ff->pff_width) {
486                                 char *cp = v +  ff->pff_width - 1;
487                                 if (ff->pff_width < 4)
488                                         die(gettext(ERR_STATS_FORMAT),
489                                             ff->pff_header);
490                                 *cp-- = 0;
491                                 *cp-- = '.';
492                                 *cp-- = '.';
493                                 *cp-- = '.';
494                         }
495                         used = snprintf(str + pos, left, "%-*s", ff->pff_width,
496                             v);
497                 }
498                 break;
499         }

501         return (used);
502 }

504 /* format big numbers */
505 static int
506 bigno_f(char *str, int pos, int left, poolstat_field_format_t *ff, char *data)
507 {
508         uint64_t v;
509         char    tag;
510         int     pw = ff->pff_width - 4;
511         double  pv;
512         int     used;

514         v = *((uint64_t *)(void *)(data + ff->pff_offset));
515         /*
516          * the max value can be ULONG_MAX, which is formatted as:
517          * E  P   T   G   M   K
518          * 18 446 744 073 709 551 615
519          * As a result ULONG_MAX is displayed as 18E
520          */
521         pv = v;
522         if (v < 1000) {
523                 pw = 0;
```

```
524         } else if (v < KILO * 10) {
525                 pv = (double)v / KILO;
526                 tag = 'K';
527         } else if (v < KILO * 100) {
528                 pv = (double)v / KILO;
529                 tag = 'K'; pw -= 1;
530         } else if (v < KILO * 1000) {
531                 pv = (double)v / KILO;
532                 tag = 'K'; pw -= 2;
533         } else if (v < MEGA * 10) {
534                 pv = (double)v / MEGA;
535                 tag = 'M';
536         } else if (v < MEGA * 100) {
537                 pv = (double)v / MEGA;
538                 tag = 'M'; pw -= 1;
539         } else if (v < MEGA * 1000) {
540                 pv = (double)v / MEGA;
541                 tag = 'M'; pw -= 2;
542         } else if (v < GIGA * 10) {
543                 pv = (double)v / GIGA;
544                 tag = 'G';
545         } else if (v < GIGA * 100) {
546                 pv = (double)v / GIGA;
547                 tag = 'G'; pw -= 1;
548         } else if (v < GIGA * 1000) {
549                 pv = (double)v / GIGA;
550                 tag = 'G'; pw -= 2;
551         } else if (v < TERA * 10) {
552                 pv = (double)v / TERA;
553                 tag = 'T';
554         } else if (v < TERA * 100) {
555                 pv = (double)v / TERA;
556                 tag = 'T'; pw -= 1;
557         } else if (v < TERA * 1000) {
558                 pv = (double)v / TERA;
559                 tag = 'T'; pw -= 2;
560         } else if (v < PETA * 10) {
561                 pv = (double)v / PETA;
562                 tag = 'P';
563         } else if (v < PETA * 100) {
564                 pv = (double)v / PETA;
565                 tag = 'P'; pw -= 1;
566         } else if (v < PETA * 1000) {
567                 pv = (double)v / PETA;
568                 tag = 'P'; pw -= 2;
569         } else if (v < EXA * 10) {
570                 pv = (double)v / EXA;
571                 tag = 'E';
572         } else if (v < EXA * 100) {
573                 pv = (double)v / EXA;
574                 tag = 'E'; pw -= 1;
575         } else {
576                 pv = (double)v / EXA;
577                 tag = 'E'; pw -= 2;
578         }
579         if (pw < 0)
580                 pw = 0;
581         if (v < 1000)
582                 used = snprintf(str + pos, left, "%*.*f",
583                     ff->pff_width, pw, pv);
584         else
585                 used = snprintf(str + pos, left, "%*.*f%c",
586                     ff->pff_width - 1, pw, pv, tag);

588         return (used);
589 }
```



```
591 /* format usage statistic, if configuration has changed print '-'. */
592 static int
593 used_stat_f(char *str, int pos, int left, poolstat_field_format_t *ff,
594         char *data)
595 {
596         int     pw = 0;
597         double v = *((double *)(void *)(data + ff->pff_offset));
598         int     used;

600         if (pool_sbag->sb_changed & POU_POOL) {
601                 used = snprintf(str + pos, left, "%*c", ff->pff_width, '-');
602         } else {
603                 if (v < 10) {
604                         pw = ff->pff_width - 2;
605                 } else if (v < 100) {
606                         pw = ff->pff_width - 3;
607                 } else if (v < 1000) {
608                         pw = ff->pff_width - 4;
609                 }
610                 if (pw < 0)
611                         pw = 0;
612                 used = snprintf(str + pos, left, "%*.*f",
613                     ff->pff_width, pw, v);
614         }
615         return (used);
616 }

618 /*
619  * Format one header field and put it into the 'str' buffer.
620  */
621 /*ARGSUSED*/
622 static int
623 header_f(char *str, int pos, int left, poolstat_field_format_t *ff, char *data)
624 {
625         int  used = 0;

627         if (ff->pff_type == STR)
628                 /* strings are left justified   */
629                 used = snprintf(str + pos, left, "%-*s",
630                     ff->pff_width, ff->pff_header);
631         else
632                 used = snprintf(str + pos, left, "%*s",
633                     ff->pff_width, ff->pff_header);
634         return (used);
635 }

637 /*
638  * Print one statistic line according to the definitions in 'lf'.
639  */
640 static void
641 prt_stat_line(poolstat_line_format_t *lf)
642 {
643         poolstat_list_element_t *le;    /* list element in the print sequence */
644         char    *line;
645         int     pos     = 0;            /* position in the printed line */
646         int     len     = MAXLINE;      /* the length of the line       */
647         int     left    = len;          /* chars left to use in the line */

649         line = ZALLOC(len);
650         for (le = lf->plf_prt_seq; le; le = le->ple_next) {
651                 int used;
652                 poolstat_field_format_t *ff =
653                     (poolstat_field_format_t *)le->ple_obj;
654                 /* if the filed is marked to be printed */
655                 if (ff->pff_prt & PABLE_FIELD) {
```

```
656                         if (((used = ff->pff_format(line, pos, left, ff,
657                             *ff->pff_data_ptr)) + 1) >= left) {
658                                 /* if field doesn't fit allocate new space */
659                                 len += used + MAXLINE;
660                                 left += used + MAXLINE;
661                                 line = REALLOC(line, len);
662                                 if (((used = ff->pff_format(line, pos, left, ff,
663                                     *ff->pff_data_ptr)) + 1) >= left)
664                                         die(gettext(ERR_STATS_FORMAT), line);
665                         }
666                         left -= used;
667                         pos += used;
668                         if (le->ple_next != NULL) {
669                                 /* separate columns with a space */
670                                 line[pos++] = ' ';
671                                 left--;
672                         }
673                 }
674         }

676         (void) printf("%s\n", line);
677         FREE(line);
678 }

680 /*
681  * Print a statistics header line for a given resource type.
682  */
683 static void
684 prt_stat_hd(const char *type)
685 {
686         poolstat_line_format_t  *lf;    /* line format  */
687         poolstat_list_element_t *le;    /* list element in the print sequence */
688         char    *line;
689         int     pos     = 0;            /* position in the printed line  */
690         int     len     = MAXLINE;      /* the length of the line        */
691         int     left    = len;          /* chars left to use in the line */

693         if (strcmp(type, POOL_TYPE_NAME) == 0) {
694                 /* pool format needs an extra header    */
695                 (void) printf("%*s\n", 19 + 15, "pset");
696                 lf = &pool_lf;
697         } else if (strcmp(type, PSET_TYPE_NAME) == 0) {
698                 lf = &pset_lf;
699         } else {
700                 die(gettext(ERR_UNSUPP_RTYPE), type);
701         }
702         line = ZALLOC(len);
703         for (le = lf->plf_prt_seq; le; le = le->ple_next) {
704                 int used;       /* used chars in line   */
705                 poolstat_field_format_t *ff =
706                     (poolstat_field_format_t *)le->ple_obj;
707                 /* if the filed is marked to be printed */
708                 if (ff->pff_prt& PABLE_FIELD) {
709                         if (((used = header_f(line, pos, left, ff, NULL)) + 1)
710                             >= left) {
711                                 /* if field doesn't fit allocate new space */
712                                 len += used + MAXLINE;
713                                 left += used + MAXLINE;
714                                 line = REALLOC(line, len);
715                                 if (((used = header_f(line, pos, left, ff,
716                                     NULL)) + 1) >= left)
717                                         die(gettext(ERR_STATS_FORMAT), line);
718                         }
719                         left -= used;
720                         pos += used;
721                         if (le->ple_next != NULL) {
```



```
722                                 /* separate columns with a space */
723                                 line[pos++] = ' ';
724                                 left--;
725                         }
726                 }
727         }

729         /* only header line with non space characters should be printed */
730         pos = 0;
731         while (*(line + pos) != '\n') {
732                 if (!isspace(*(line + pos))) {
733                         (void) printf("%s\n", line);

735                         break;
736                 }
737                 pos++;
738         }
739         FREE(line);
740 }

742 /*
743  * Create a pool value instance and set its name to 'name'.
744  */
745 static pool_value_t *
746 create_pool_value(const char *name)
747 {
748         pool_value_t *pval;

750         if ((pval = pool_value_alloc()) == NULL) {
751                 return (NULL);
752         }
753         if (pool_value_set_name(pval, name) != PO_SUCCESS) {
754                 pool_value_free(pval);
755                 return (NULL);
756         }

758         return (pval);
759 }

761 /*
762  * Find all resources of type 'rtype'.
763  * If 'pool_name' is defined find all resources bound to this pool.
764  */
765 static pool_resource_t **
766 get_resources(const char *pool_name, const char *rtype, uint_t *nelem)
767 {
768         pool_resource_t **resources = NULL;
769         pool_value_t    *pvals[] = { NULL, NULL, NULL};
770         pool_value_t    *pv_sys_id;
771         pool_value_t    *pv_name;
772         char            *name_prop; /* set name property        */

774         if (strcmp(rtype, PSET_TYPE_NAME) == 0) {
775                 if ((pv_sys_id = create_pool_value(PSET_SYSID)) == NULL)
776                         goto on_error;
777                 name_prop = PSET_NAME;
778         } else {
779                 die(gettext(ERR_UNSUPP_RTYPE), rtype);
780         }

782         if ((pvals[0] = create_pool_value("type")) == NULL)
783                 goto on_error;
784         if ((pool_value_set_string(pvals[0], rtype)) == -1)
785                 goto on_error;

787         if ((pv_name = create_pool_value(name_prop)) == NULL)
```

```
788                 goto on_error;

790         if (pool_name != NULL) {
791                 /* collect resources associated to 'pool_name'  */
792                 pool_t  *pool;
793                 if ((pool = pool_get_pool(conf, pool_name)) == NULL)
794                         die(gettext(ERR_STATS_POOL_N), pool_name);
795                 if ((resources = pool_query_pool_resources(
796                     conf, pool, nelem, pvals)) == NULL)
797                         goto on_error;
798         } else {
799                 /* collect all resources  */
800                 if ((resources =
801                     pool_query_resources(conf, nelem, pvals)) == NULL)
802                         goto on_error;
803         }

805         if (pv_name != NULL)
806                 pool_value_free(pv_name);
807         if (pv_sys_id != NULL)
808                 pool_value_free(pv_sys_id);
809         if (pvals[0] != NULL)
810                 pool_value_free(pvals[0]);

812         return (resources);
813 on_error:
814         die(gettext(ERR_STATS_RES), get_errstr());
815         /*NOTREACHED*/
816 }

818 /*
819  * Print statistics for all resources of type 'rtype' passed in 'resources'.
820  */
821 static void
822 prt_resource_stats_by_type(pool_resource_t **resources, const char *rtype)
823 {
824         int             i;
825         pool_elem_t     *elem;
826         pool_value_t    *pv_name;
827         char            *name_prop;

829         poolstat_line_format_t  *lf;
830         statistic_bag_t         *sbag;

832         if (strcmp(rtype, PSET_TYPE_NAME) == 0) {
833                 name_prop = PSET_NAME;
834                 lf = &pset_lf;
835                 sbag = pset_sbag;
836         } else {
837                 die(gettext(ERR_UNSUPP_RTYPE), rtype);
838         }

840         if ((pv_name = create_pool_value(name_prop)) == NULL)
841                 goto on_error;

843         /* collect and print statistics for the given resources */
844         for (i = 0; resources[i] != NULL; i++) {
845                 if ((elem = pool_resource_to_elem(conf, resources[i])) == NULL)
846                         goto on_error;
847                 if (pool_get_property(conf, elem, name_prop, pv_name) == -1)
848                         goto on_error;
849                 if (pool_value_get_string(pv_name, &sbag->sb_name) == -1)
850                         goto on_error;
851                 sa_update(sbag, 0);

853                 prt_stat_line(lf);
```



```
854         }

856         if (pv_name != NULL)
857                 pool_value_free(pv_name);
858         return;
859 on_error:
860         die(gettext(ERR_STATS_RES), get_errstr());
861 }

863 /*
864  * Update statistics for all resources of type 'rtype' pased in 'resources'.
865  */
866 static void
867 update_resource_stats(pool_resource_t *resource, const char *rtype)
868 {
869         pool_elem_t     *elem;
870         pool_value_t    *pv_name;
871         char            *name_prop;             /* set name property    */

873         statistic_bag_t *sbag;

875         if (strcmp(rtype, PSET_TYPE_NAME) == 0) {
876                 name_prop = PSET_NAME;
877                 sbag    = pset_sbag;
878         } else {
879                 die(gettext(ERR_UNSUPP_RTYPE), rtype);
880         }

882         if ((pv_name = create_pool_value(name_prop)) == NULL)
883                 goto on_error;

885         if ((elem = pool_resource_to_elem(conf, resource)) == NULL)
886                 goto on_error;
887         if (pool_get_property(conf, elem, name_prop, pv_name) == -1)
888                 goto on_error;
889         if (pool_value_get_string(pv_name, &sbag->sb_name) == -1)
890                 goto on_error;
891         sa_update(sbag, 0);

893         if (pv_name != NULL)
894                 pool_value_free(pv_name);
895         return;

897 on_error:
898         die(gettext(ERR_STATS_RES), get_errstr());
899 }

901 /*
902  * For each pool in the configuration print statistics of associated resources.
903  * If the pool name list 'pn' is defined, only print resources of pools
904  * specified in the list. The list can specify the pool name or its system id.
905  */
906 static void
907 prt_pool_stats(poolstat_list_element_t *pn)
908 {
909         uint_t          nelem;
910         pool_elem_t     *elem;
911         int             i;
912         int             error;
913         pool_t          **pools = NULL;
914         pool_value_t    *pvals[] = { NULL, NULL };
915         pool_value_t    *pv_name = NULL;
916         pool_value_t    *pv_sys_id = NULL;
917         statistic_bag_t *sbag = pool_sbag;
918         poolstat_list_element_t         *rtype;
919         pool_resource_t **resources;
```

```
921         if ((pv_sys_id = create_pool_value(POOL_SYSID)) == NULL)
922                 goto on_error;
923         if ((pv_name = create_pool_value(POOL_NAME)) == NULL)
924                 goto on_error;

926         if (pn == NULL) {
927                 /* collect all pools    */
928                 if ((pools = pool_query_pools(conf, &nelem, NULL)) == NULL)
929                         goto on_error;
930         } else {
931                 /*
932                  * collect pools specified in the 'pn' list.
933                  * 'poolid' the pool identifier can be a pool name or sys_id.
934                  */
935                 poolstat_list_element_t *poolid;
936                 for (poolid = pn, i = 1; poolid; poolid = poolid->ple_next)
937                         i++;
938                 pools = ZALLOC(sizeof (pool_t *) * (i + 1));
939                 for (poolid = pn, i = 0; poolid;
940                     poolid = poolid->ple_next, i++) {
941                         pool_t **pool;
942                         int64_t sysid = Atoi(poolid->ple_obj, &error);
943                         if (error == 0) {
944                                 /* the pool is identified by sys_id     */
945                                 pool_value_set_int64(pv_sys_id, sysid);
946                                 pvals[0] = pv_sys_id;
947                                 pool = pool_query_pools(conf, &nelem, pvals);
948                         } else {
949                                 if (pool_value_set_string(pv_name,
950                                     poolid->ple_obj) == -1)
951                                         die(gettext(ERR_NOMEM));
952                                 pvals[0] = pv_name;
953                                 pool = pool_query_pools(conf, &nelem, pvals);
954                         }
955                         if (pool == NULL)
956                                 die(gettext(ERR_STATS_POOL_N), poolid->ple_obj);
957                         pools[i] = pool[0];
958                         FREE(pool);
959                 }
960         }

962         /* print statistic for all pools found          */
963         if (!rflag) {
964                 /* print the common resource header     */
965                 prt_stat_hd(POOL_TYPE_NAME);

967                 /* print statistics for the resources bound to the pools */
968                 for (i = 0; pools[i] != NULL; i++) {
969                         elem = pool_to_elem(conf, pools[i]);
970                         if (pool_get_property(conf, elem, POOL_NAME, pv_name)
971                             == -1)
972                                 goto on_error;
973                         if (pool_value_get_string(pv_name, &sbag->sb_name) != 0)
974                                 goto on_error;
975                         if (pool_get_property(
976                             conf, elem, "pool.sys_id", pv_sys_id) == -1)
977                                 goto on_error;
978                         if (pool_value_get_int64(
979                             pv_sys_id, &sbag->sb_sysid) != 0)
980                                 goto on_error;

982                         for (rtype = rtypes; rtype; rtype = rtype->ple_next) {
983                                 resources = get_resources(
984                                     sbag->sb_name, rtype->ple_obj, &nelem);
985                                 update_resource_stats(*resources,
```



```
986                                     rtype->ple_obj);
987                                 FREE(resources);
988                         }
989                         prt_stat_line(&pool_lf);
990                 }
991         } else {
992                 /* print statistic for all resource types defined in rtypes */
993                 for (rtype = rtypes; rtype; rtype = rtype->ple_next) {
994                         prt_stat_hd(rtype->ple_obj);
995                         for (i = 0; pools[i] != NULL; i++) {
996                                 elem = pool_to_elem(conf, pools[i]);
997                                 if (pool_get_property(
998                                     conf, elem, POOL_NAME, pv_name) == -1)
999                                         goto on_error;
1000                                if (pool_value_get_string(
1001                                    pv_name, &sbag->sb_name) != 0)
1002                                        goto on_error;
1003                                if (pool_get_property(
1004                                    conf, elem, POOL_SYSID, pv_sys_id) == -1)
1005                                        goto on_error;
1006                                if (pool_value_get_int64(
1007                                    pv_sys_id, &sbag->sb_sysid) != 0)
1008                                        goto on_error;
1009                                resources = get_resources(
1010                                    sbag->sb_name, rtype->ple_obj, &nelem);
1011                                if (resources == NULL)
1012                                        continue;
1013                                update_resource_stats(
1014                                    *resources, rtype->ple_obj);
1015                                prt_resource_stats_by_type(resources,
1016                                    rtype->ple_obj);
1017                                FREE(resources);
1018                        }
1019                }
1020        }

1022        FREE(pools);
1023        if (pv_name != NULL)
1024                pool_value_free(pv_name);
1025        if (pv_sys_id != NULL)
1026                pool_value_free(pv_sys_id);

1028        return;
1029 on_error:
1030        die(gettext(ERR_STATS_POOL), get_errstr());
1031 }
```

****
26949 Thu Sep 5 23:02:06 2013
new/usr/src/cmd/stat/common/acquire_iodevs.c
3373 gcc >= 4.5 concerns about offsetof()
****
_____***unchanged_portion_omitted_***

```
 145 /*
 146  * Introduce an index into the list to speed up insert_into looking for the
 147  * right position in the list. This index is an AVL tree of all the
 148  * iodev_snapshot in the list.
 149  */

 151 #if defined(__GNUC__)
 152 #define offsetof(s, m)  __builtin_offsetof(s, m)
 153 #else
 154 #define offsetof(s, m)  ((size_t)(&(((s *)0)->m)))      /* for avl_create */
 155 #endif
 151 #define offsetof(s, m)  (size_t)(&(((s *)0)->m))        /* for avl_create */

 157 static int
 158 avl_iodev_cmp(const void* is1, const void* is2)
 159 {
 160         int c = iodev_cmp((struct iodev_snapshot *)is1,
 161             (struct iodev_snapshot *)is2);

 163         if (c > 0)
 164                 return (1);

 166         if (c < 0)
 167                 return (-1);

 169         return (0);
 170 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   75157 Thu Sep  5 23:02:07 2013
new/usr/src/common/nvpair/nvpair.c
3373 gcc >= 4.5 concerns about offsetof()
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
 */

#include <sys/stropts.h>
#include <sys/debug.h>
#include <sys/isa_defs.h>
#include <sys/int_limits.h>
#include <sys/nvpair.h>
#include <sys/nvpair_impl.h>
#include <rpc/types.h>
#include <rpc/xdr.h>

#if defined(_KERNEL) && !defined(_BOOT)
#include <sys/varargs.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#else
#include <stdarg.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#endif

#ifndef offsetof
#if defined(__GNUC__)
#define offsetof(s, m)  __builtin_offsetof(s, m)
#else
#endif /* ! codereview */
#define offsetof(s, m)  ((size_t)(&(((s *)0)->m)))
#endif
#endif /* ! codereview */
#endif
#define skip_whitespace(p)      while ((*(p) == ' ') || (*(p) == '\t')) p++

/*
 * nvpair.c - Provides kernel & userland interfaces for manipulating
 *      name-value pairs.
 *
 * Overview Diagram
```



```
 *
 *  +--------------+
 *  |  nvlist_t    |
 *  |--------------|
 *  | nvl_version  |
 *  | nvl_nvflag   |
 *  | nvl_priv    -+-+
 *  | nvl_flag     | |
 *  | nvl_pad      | |
 *  +--------------+ |
 *                   V
 *      +--------------+      last i_nvp in list
 *      | nvpriv_t     |  +--------------------->
 *      |--------------|  |
 *   +--+- nvp_list    |  |   +------------+
 *   |  |  nvp_last   -+--+   + nv_alloc_t |
 *   |  |  nvp_curr    |      |------------|
 *   |  |  nvp_nva    -+----> | nva_ops    |
 *   |  |  nvp_stat    |      | nva_arg    |
 *   |  +--------------+      +------------+
 *   |
 *   +-------+
 *           V
 *   +---------------------+      +-------------------+
 *   |  i_nvp_t            |  +-->|  i_nvp_t          |  +-->
 *   |---------------------|  |   |-------------------|  |
 *   | nvi_next           -+--+   | nvi_next         -+--+
 *   | nvi_prev (NULL)     | <----+ nvi_prev          |
 *   | . . . . . . . . . . |      | . . . . . . . . . . |
 *   | nvp (nvpair_t)      |      | nvp (nvpair_t)    |
 *   |  - nvp_size         |      |  - nvp_size       |
 *   |  - nvp_name_sz      |      |  - nvp_name_sz    |
 *   |  - nvp_value_elem   |      |  - nvp_value_elem |
 *   |  - nvp_type         |      |  - nvp_type       |
 *   |  - data ...         |      |  - data ...       |
 *   +---------------------+      +-------------------+
 *
 *
 *
 *   +---------------------+             +---------------------+
 *   |  i_nvp_t            |  +-->    +-->|  i_nvp_t (last)     |
 *   |---------------------|  |      |   |---------------------|
 *   |  nvi_next          -+--+ ... --+   | nvi_next (NULL)     |
 * <-+- nvi_prev           |<-- ...  <----+ nvi_prev            |
 *   | . . . . . . . . . . |             | . . . . . . . . . . |
 *   | nvp (nvpair_t)      |             | nvp (nvpair_t)      |
 *   |  - nvp_size         |             |  - nvp_size         |
 *   |  - nvp_name_sz      |             |  - nvp_name_sz      |
 *   |  - nvp_value_elem   |             |  - nvp_value_elem   |
 *   |  - DATA_TYPE_NVLIST |             |  - nvp_type         |
 *   |  - data (embedded)  |             |  - data ...         |
 *   |    nvlist name      |             +---------------------+
 *   |  +--------------+   |
 *   |  |  nvlist_t    |   |
 *   |  |--------------|   |
 *   |  | nvl_version  |   |
 *   |  | nvl_nvflag   |   |
 *   |  | nvl_priv   --+---+---->
 *   |  | nvl_flag     |   |
 *   |  | nvl_pad      |   |
 *   |  +--------------+   |
 *   +---------------------+
 *
 *
 * N.B. nvpair_t may be aligned on 4 byte boundary, so +4 will
 * allow value to be aligned on 8 byte boundary
```

```
 *
 * name_len is the length of the name string including the null terminator
 * so it must be >= 1
 */
#define NVP_SIZE_CALC(name_len, data_len) \
        (NV_ALIGN((sizeof (nvpair_t)) + name_len) + NV_ALIGN(data_len))

static int i_get_value_size(data_type_t type, const void *data, uint_t nelem);
static int nvlist_add_common(nvlist_t *nvl, const char *name, data_type_t type,
    uint_t nelem, const void *data);

#define NV_STAT_EMBEDDED        0x1
#define EMBEDDED_NVL(nvp)       ((nvlist_t *)(void *)NVP_VALUE(nvp))
#define EMBEDDED_NVL_ARRAY(nvp) ((nvlist_t **)(void *)NVP_VALUE(nvp))

#define NVP_VALOFF(nvp) (NV_ALIGN(sizeof (nvpair_t) + (nvp)->nvp_name_sz))
#define NVPAIR2I_NVP(nvp) \
        ((i_nvp_t *)((size_t)(nvp) - offsetof(i_nvp_t, nvi_nvp)))


int
nv_alloc_init(nv_alloc_t *nva, const nv_alloc_ops_t *nvo, /* args */ ...)
{
        va_list valist;
        int err = 0;

        nva->nva_ops = nvo;
        nva->nva_arg = NULL;

        va_start(valist, nvo);
        if (nva->nva_ops->nv_ao_init != NULL)
                err = nva->nva_ops->nv_ao_init(nva, valist);
        va_end(valist);

        return (err);
}

void
nv_alloc_reset(nv_alloc_t *nva)
{
        if (nva->nva_ops->nv_ao_reset != NULL)
                nva->nva_ops->nv_ao_reset(nva);
}

void
nv_alloc_fini(nv_alloc_t *nva)
{
        if (nva->nva_ops->nv_ao_fini != NULL)
                nva->nva_ops->nv_ao_fini(nva);
}

nv_alloc_t *
nvlist_lookup_nv_alloc(nvlist_t *nvl)
{
        nvpriv_t *priv;

        if (nvl == NULL ||
            (priv = (nvpriv_t *)(uintptr_t)nvl->nvl_priv) == NULL)
                return (NULL);

        return (priv->nvp_nva);
}

static void *
nv_mem_zalloc(nvpriv_t *nvp, size_t size)
{
```



```
        nv_alloc_t *nva = nvp->nvp_nva;
        void *buf;

        if ((buf = nva->nva_ops->nv_ao_alloc(nva, size)) != NULL)
                bzero(buf, size);

        return (buf);
}

static void
nv_mem_free(nvpriv_t *nvp, void *buf, size_t size)
{
        nv_alloc_t *nva = nvp->nvp_nva;

        nva->nva_ops->nv_ao_free(nva, buf, size);
}

static void
nv_priv_init(nvpriv_t *priv, nv_alloc_t *nva, uint32_t stat)
{
        bzero(priv, sizeof (nvpriv_t));

        priv->nvp_nva = nva;
        priv->nvp_stat = stat;
}

static nvpriv_t *
nv_priv_alloc(nv_alloc_t *nva)
{
        nvpriv_t *priv;

        /*
         * nv_mem_alloc() cannot called here because it needs the priv
         * argument.
         */
        if ((priv = nva->nva_ops->nv_ao_alloc(nva, sizeof (nvpriv_t))) == NULL)
                return (NULL);

        nv_priv_init(priv, nva, 0);

        return (priv);
}

/*
 * Embedded lists need their own nvpriv_t's.  We create a new
 * nvpriv_t using the parameters and allocator from the parent
 * list's nvpriv_t.
 */
static nvpriv_t *
nv_priv_alloc_embedded(nvpriv_t *priv)
{
        nvpriv_t *emb_priv;

        if ((emb_priv = nv_mem_zalloc(priv, sizeof (nvpriv_t))) == NULL)
                return (NULL);

        nv_priv_init(emb_priv, priv->nvp_nva, NV_STAT_EMBEDDED);

        return (emb_priv);
}

static void
nvlist_init(nvlist_t *nvl, uint32_t nvflag, nvpriv_t *priv)
{
        nvl->nvl_version = NV_VERSION;
        nvl->nvl_nvflag = nvflag & (NV_UNIQUE_NAME|NV_UNIQUE_NAME_TYPE);
```

```
260         nvl->nvl_priv = (uint64_t)(uintptr_t)priv;
261         nvl->nvl_flag = 0;
262         nvl->nvl_pad = 0;
263 }

265 uint_t
266 nvlist_nvflag(nvlist_t *nvl)
267 {
268         return (nvl->nvl_nvflag);
269 }

271 /*
272  * nvlist_alloc - Allocate nvlist.
273  */
274 /*ARGSUSED1*/
275 int
276 nvlist_alloc(nvlist_t **nvlp, uint_t nvflag, int kmflag)
277 {
278 #if defined(_KERNEL) && !defined(_BOOT)
279         return (nvlist_xalloc(nvlp, nvflag,
280             (kmflag == KM_SLEEP ? nv_alloc_sleep : nv_alloc_nosleep)));
281 #else
282         return (nvlist_xalloc(nvlp, nvflag, nv_alloc_nosleep));
283 #endif
284 }

286 int
287 nvlist_xalloc(nvlist_t **nvlp, uint_t nvflag, nv_alloc_t *nva)
288 {
289         nvpriv_t *priv;

291         if (nvlp == NULL || nva == NULL)
292                 return (EINVAL);

294         if ((priv = nv_priv_alloc(nva)) == NULL)
295                 return (ENOMEM);

297         if ((*nvlp = nv_mem_zalloc(priv,
298             NV_ALIGN(sizeof (nvlist_t)))) == NULL) {
299                 nv_mem_free(priv, priv, sizeof (nvpriv_t));
300                 return (ENOMEM);
301         }

303         nvlist_init(*nvlp, nvflag, priv);

305         return (0);
306 }

308 /*
309  * nvp_buf_alloc - Allocate i_nvp_t for storing a new nv pair.
310  */
311 static nvpair_t *
312 nvp_buf_alloc(nvlist_t *nvl, size_t len)
313 {
314         nvpriv_t *priv = (nvpriv_t *)(uintptr_t)nvl->nvl_priv;
315         i_nvp_t *buf;
316         nvpair_t *nvp;
317         size_t nvsize;

319         /*
320          * Allocate the buffer
321          */
322         nvsize = len + offsetof(i_nvp_t, nvi_nvp);

324         if ((buf = nv_mem_zalloc(priv, nvsize)) == NULL)
325                 return (NULL);
```



```
327         nvp = &buf->nvi_nvp;
328         nvp->nvp_size = len;

330         return (nvp);
331 }

333 /*
334  * nvp_buf_free - de-Allocate an i_nvp_t.
335  */
336 static void
337 nvp_buf_free(nvlist_t *nvl, nvpair_t *nvp)
338 {
339         nvpriv_t *priv = (nvpriv_t *)(uintptr_t)nvl->nvl_priv;
340         size_t nvsize = nvp->nvp_size + offsetof(i_nvp_t, nvi_nvp);

342         nv_mem_free(priv, NVPAIR2I_NVP(nvp), nvsize);
343 }

345 /*
346  * nvp_buf_link - link a new nv pair into the nvlist.
347  */
348 static void
349 nvp_buf_link(nvlist_t *nvl, nvpair_t *nvp)
350 {
351         nvpriv_t *priv = (nvpriv_t *)(uintptr_t)nvl->nvl_priv;
352         i_nvp_t *curr = NVPAIR2I_NVP(nvp);

354         /* Put element at end of nvlist */
355         if (priv->nvp_list == NULL) {
356                 priv->nvp_list = priv->nvp_last = curr;
357         } else {
358                 curr->nvi_prev = priv->nvp_last;
359                 priv->nvp_last->nvi_next = curr;
360                 priv->nvp_last = curr;
361         }
362 }

364 /*
365  * nvp_buf_unlink - unlink an removed nvpair out of the nvlist.
366  */
367 static void
368 nvp_buf_unlink(nvlist_t *nvl, nvpair_t *nvp)
369 {
370         nvpriv_t *priv = (nvpriv_t *)(uintptr_t)nvl->nvl_priv;
371         i_nvp_t *curr = NVPAIR2I_NVP(nvp);

373         /*
374          * protect nvlist_next_nvpair() against walking on freed memory.
375          */
376         if (priv->nvp_curr == curr)
377                 priv->nvp_curr = curr->nvi_next;

379         if (curr == priv->nvp_list)
380                 priv->nvp_list = curr->nvi_next;
381         else
382                 curr->nvi_prev->nvi_next = curr->nvi_next;

384         if (curr == priv->nvp_last)
385                 priv->nvp_last = curr->nvi_prev;
386         else
387                 curr->nvi_next->nvi_prev = curr->nvi_prev;
388 }

390 /*
391  * take a nvpair type and number of elements and make sure the are valid
```

```
 */
static int
i_validate_type_nelem(data_type_t type, uint_t nelem)
{
        switch (type) {
        case DATA_TYPE_BOOLEAN:
                if (nelem != 0)
                        return (EINVAL);
                break;
        case DATA_TYPE_BOOLEAN_VALUE:
        case DATA_TYPE_BYTE:
        case DATA_TYPE_INT8:
        case DATA_TYPE_UINT8:
        case DATA_TYPE_INT16:
        case DATA_TYPE_UINT16:
        case DATA_TYPE_INT32:
        case DATA_TYPE_UINT32:
        case DATA_TYPE_INT64:
        case DATA_TYPE_UINT64:
        case DATA_TYPE_STRING:
        case DATA_TYPE_HRTIME:
        case DATA_TYPE_NVLIST:
#if !defined(_KERNEL)
        case DATA_TYPE_DOUBLE:
#endif
                if (nelem != 1)
                        return (EINVAL);
                break;
        case DATA_TYPE_BOOLEAN_ARRAY:
        case DATA_TYPE_BYTE_ARRAY:
        case DATA_TYPE_INT8_ARRAY:
        case DATA_TYPE_UINT8_ARRAY:
        case DATA_TYPE_INT16_ARRAY:
        case DATA_TYPE_UINT16_ARRAY:
        case DATA_TYPE_INT32_ARRAY:
        case DATA_TYPE_UINT32_ARRAY:
        case DATA_TYPE_INT64_ARRAY:
        case DATA_TYPE_UINT64_ARRAY:
        case DATA_TYPE_STRING_ARRAY:
        case DATA_TYPE_NVLIST_ARRAY:
                /* we allow arrays with 0 elements */
                break;
        default:
                return (EINVAL);
        }
        return (0);
}

/*
 * Verify nvp_name_sz and check the name string length.
 */
static int
i_validate_nvpair_name(nvpair_t *nvp)
{
        if ((nvp->nvp_name_sz <= 0) ||
            (nvp->nvp_size < NVP_SIZE_CALC(nvp->nvp_name_sz, 0)))
                return (EFAULT);

        /* verify the name string, make sure its terminated */
        if (NVP_NAME(nvp)[nvp->nvp_name_sz - 1] != '\0')
                return (EFAULT);

        return (strlen(NVP_NAME(nvp)) == nvp->nvp_name_sz - 1 ? 0 : EFAULT);
}

static int
```



```
i_validate_nvpair_value(data_type_t type, uint_t nelem, const void *data)
{
        switch (type) {
        case DATA_TYPE_BOOLEAN_VALUE:
                if (*(boolean_t *)data != B_TRUE &&
                    *(boolean_t *)data != B_FALSE)
                        return (EINVAL);
                break;
        case DATA_TYPE_BOOLEAN_ARRAY: {
                int i;

                for (i = 0; i < nelem; i++)
                        if (((boolean_t *)data)[i] != B_TRUE &&
                            ((boolean_t *)data)[i] != B_FALSE)
                                return (EINVAL);
                break;
        }
        default:
                break;
        }

        return (0);
}

/*
 * This function takes a pointer to what should be a nvpair and it's size
 * and then verifies that all the nvpair fields make sense and can be
 * trusted.  This function is used when decoding packed nvpairs.
 */
static int
i_validate_nvpair(nvpair_t *nvp)
{
        data_type_t type = NVP_TYPE(nvp);
        int size1, size2;

        /* verify nvp_name_sz, check the name string length */
        if (i_validate_nvpair_name(nvp) != 0)
                return (EFAULT);

        if (i_validate_nvpair_value(type, NVP_NELEM(nvp), NVP_VALUE(nvp)) != 0)
                return (EFAULT);

        /*
         * verify nvp_type, nvp_value_elem, and also possibly
         * verify string values and get the value size.
         */
        size2 = i_get_value_size(type, NVP_VALUE(nvp), NVP_NELEM(nvp));
        size1 = nvp->nvp_size - NVP_VALOFF(nvp);
        if (size2 < 0 || size1 != NV_ALIGN(size2))
                return (EFAULT);

        return (0);
}

static int
nvlist_copy_pairs(nvlist_t *snvl, nvlist_t *dnvl)
{
        nvpriv_t *priv;
        i_nvp_t *curr;

        if ((priv = (nvpriv_t *)(uintptr_t)snvl->nvl_priv) == NULL)
                return (EINVAL);

        for (curr = priv->nvp_list; curr != NULL; curr = curr->nvi_next) {
                nvpair_t *nvp = &curr->nvi_nvp;
                int err;
```

```
525                 if ((err = nvlist_add_common(dnvl, NVP_NAME(nvp), NVP_TYPE(nvp),
526                     NVP_NELEM(nvp), NVP_VALUE(nvp))) != 0)
527                         return (err);
528         }

530         return (0);
531 }

533 /*
534  * Frees all memory allocated for an nvpair (like embedded lists) with
535  * the exception of the nvpair buffer itself.
536  */
537 static void
538 nvpair_free(nvpair_t *nvp)
539 {
540         switch (NVP_TYPE(nvp)) {
541         case DATA_TYPE_NVLIST:
542                 nvlist_free(EMBEDDED_NVL(nvp));
543                 break;
544         case DATA_TYPE_NVLIST_ARRAY: {
545                 nvlist_t **nvlp = EMBEDDED_NVL_ARRAY(nvp);
546                 int i;

548                 for (i = 0; i < NVP_NELEM(nvp); i++)
549                         if (nvlp[i] != NULL)
550                                 nvlist_free(nvlp[i]);
551                 break;
552         }
553         default:
554                 break;
555         }
556 }

558 /*
559  * nvlist_free - free an unpacked nvlist
560  */
561 void
562 nvlist_free(nvlist_t *nvl)
563 {
564         nvpriv_t *priv;
565         i_nvp_t *curr;

567         if (nvl == NULL ||
568             (priv = (nvpriv_t *)(uintptr_t)nvl->nvl_priv) == NULL)
569                 return;

571         /*
572          * Unpacked nvlist are linked through i_nvp_t
573          */
574         curr = priv->nvp_list;
575         while (curr != NULL) {
576                 nvpair_t *nvp = &curr->nvi_nvp;
577                 curr = curr->nvi_next;

579                 nvpair_free(nvp);
580                 nvp_buf_free(nvl, nvp);
581         }

583         if (!(priv->nvp_stat & NV_STAT_EMBEDDED))
584                 nv_mem_free(priv, nvl, NV_ALIGN(sizeof (nvlist_t)));
585         else
586                 nvl->nvl_priv = 0;

588         nv_mem_free(priv, priv, sizeof (nvpriv_t));
589 }
```



```
591 static int
592 nvlist_contains_nvp(nvlist_t *nvl, nvpair_t *nvp)
593 {
594         nvpriv_t *priv = (nvpriv_t *)(uintptr_t)nvl->nvl_priv;
595         i_nvp_t *curr;

597         if (nvp == NULL)
598                 return (0);

600         for (curr = priv->nvp_list; curr != NULL; curr = curr->nvi_next)
601                 if (&curr->nvi_nvp == nvp)
602                         return (1);

604         return (0);
605 }

607 /*
608  * Make a copy of nvlist
609  */
610 /*ARGSUSED1*/
611 int
612 nvlist_dup(nvlist_t *nvl, nvlist_t **nvlp, int kmflag)
613 {
614 #if defined(_KERNEL) && !defined(_BOOT)
615         return (nvlist_xdup(nvl, nvlp,
616             (kmflag == KM_SLEEP ? nv_alloc_sleep : nv_alloc_nosleep)));
617 #else
618         return (nvlist_xdup(nvl, nvlp, nv_alloc_nosleep));
619 #endif
620 }

622 int
623 nvlist_xdup(nvlist_t *nvl, nvlist_t **nvlp, nv_alloc_t *nva)
624 {
625         int err;
626         nvlist_t *ret;

628         if (nvl == NULL || nvlp == NULL)
629                 return (EINVAL);

631         if ((err = nvlist_xalloc(&ret, nvl->nvl_nvflag, nva)) != 0)
632                 return (err);

634         if ((err = nvlist_copy_pairs(nvl, ret)) != 0)
635                 nvlist_free(ret);
636         else
637                 *nvlp = ret;

639         return (err);
640 }

642 /*
643  * Remove all with matching name
644  */
645 int
646 nvlist_remove_all(nvlist_t *nvl, const char *name)
647 {
648         nvpriv_t *priv;
649         i_nvp_t *curr;
650         int error = ENOENT;

652         if (nvl == NULL || name == NULL ||
653             (priv = (nvpriv_t *)(uintptr_t)nvl->nvl_priv) == NULL)
654                 return (EINVAL);
```

```
656         curr = priv->nvp_list;
657         while (curr != NULL) {
658                 nvpair_t *nvp = &curr->nvi_nvp;

660                 curr = curr->nvi_next;
661                 if (strcmp(name, NVP_NAME(nvp)) != 0)
662                         continue;

664                 nvp_buf_unlink(nvl, nvp);
665                 nvpair_free(nvp);
666                 nvp_buf_free(nvl, nvp);

668                 error = 0;
669         }

671         return (error);
672 }

674 /*
675  * Remove first one with matching name and type
676  */
677 int
678 nvlist_remove(nvlist_t *nvl, const char *name, data_type_t type)
679 {
680         nvpriv_t *priv;
681         i_nvp_t *curr;

683         if (nvl == NULL || name == NULL ||
684             (priv = (nvpriv_t *)(uintptr_t)nvl->nvl_priv) == NULL)
685                 return (EINVAL);

687         curr = priv->nvp_list;
688         while (curr != NULL) {
689                 nvpair_t *nvp = &curr->nvi_nvp;

691                 if (strcmp(name, NVP_NAME(nvp)) == 0 && NVP_TYPE(nvp) == type) {
692                         nvp_buf_unlink(nvl, nvp);
693                         nvpair_free(nvp);
694                         nvp_buf_free(nvl, nvp);

696                         return (0);
697                 }
698                 curr = curr->nvi_next;
699         }

701         return (ENOENT);
702 }

704 int
705 nvlist_remove_nvpair(nvlist_t *nvl, nvpair_t *nvp)
706 {
707         if (nvl == NULL || nvp == NULL)
708                 return (EINVAL);

710         nvp_buf_unlink(nvl, nvp);
711         nvpair_free(nvp);
712         nvp_buf_free(nvl, nvp);
713         return (0);
714 }

716 /*
717  * This function calculates the size of an nvpair value.
718  *
719  * The data argument controls the behavior in case of the data types
720  *      DATA_TYPE_STRING        and
721  *      DATA_TYPE_STRING_ARRAY
```



```
722  * Is data == NULL then the size of the string(s) is excluded.
723  */
724 static int
725 i_get_value_size(data_type_t type, const void *data, uint_t nelem)
726 {
727         uint64_t value_sz;

729         if (i_validate_type_nelem(type, nelem) != 0)
730                 return (-1);

732         /* Calculate required size for holding value */
733         switch (type) {
734         case DATA_TYPE_BOOLEAN:
735                 value_sz = 0;
736                 break;
737         case DATA_TYPE_BOOLEAN_VALUE:
738                 value_sz = sizeof (boolean_t);
739                 break;
740         case DATA_TYPE_BYTE:
741                 value_sz = sizeof (uchar_t);
742                 break;
743         case DATA_TYPE_INT8:
744                 value_sz = sizeof (int8_t);
745                 break;
746         case DATA_TYPE_UINT8:
747                 value_sz = sizeof (uint8_t);
748                 break;
749         case DATA_TYPE_INT16:
750                 value_sz = sizeof (int16_t);
751                 break;
752         case DATA_TYPE_UINT16:
753                 value_sz = sizeof (uint16_t);
754                 break;
755         case DATA_TYPE_INT32:
756                 value_sz = sizeof (int32_t);
757                 break;
758         case DATA_TYPE_UINT32:
759                 value_sz = sizeof (uint32_t);
760                 break;
761         case DATA_TYPE_INT64:
762                 value_sz = sizeof (int64_t);
763                 break;
764         case DATA_TYPE_UINT64:
765                 value_sz = sizeof (uint64_t);
766                 break;
767 #if !defined(_KERNEL)
768         case DATA_TYPE_DOUBLE:
769                 value_sz = sizeof (double);
770                 break;
771 #endif
772         case DATA_TYPE_STRING:
773                 if (data == NULL)
774                         value_sz = 0;
775                 else
776                         value_sz = strlen(data) + 1;
777                 break;
778         case DATA_TYPE_BOOLEAN_ARRAY:
779                 value_sz = (uint64_t)nelem * sizeof (boolean_t);
780                 break;
781         case DATA_TYPE_BYTE_ARRAY:
782                 value_sz = (uint64_t)nelem * sizeof (uchar_t);
783                 break;
784         case DATA_TYPE_INT8_ARRAY:
785                 value_sz = (uint64_t)nelem * sizeof (int8_t);
786                 break;
787         case DATA_TYPE_UINT8_ARRAY:
```

```
788                 value_sz = (uint64_t)nelem * sizeof (uint8_t);
789                 break;
790         case DATA_TYPE_INT16_ARRAY:
791                 value_sz = (uint64_t)nelem * sizeof (int16_t);
792                 break;
793         case DATA_TYPE_UINT16_ARRAY:
794                 value_sz = (uint64_t)nelem * sizeof (uint16_t);
795                 break;
796         case DATA_TYPE_INT32_ARRAY:
797                 value_sz = (uint64_t)nelem * sizeof (int32_t);
798                 break;
799         case DATA_TYPE_UINT32_ARRAY:
800                 value_sz = (uint64_t)nelem * sizeof (uint32_t);
801                 break;
802         case DATA_TYPE_INT64_ARRAY:
803                 value_sz = (uint64_t)nelem * sizeof (int64_t);
804                 break;
805         case DATA_TYPE_UINT64_ARRAY:
806                 value_sz = (uint64_t)nelem * sizeof (uint64_t);
807                 break;
808         case DATA_TYPE_STRING_ARRAY:
809                 value_sz = (uint64_t)nelem * sizeof (uint64_t);

811                 if (data != NULL) {
812                         char *const *strs = data;
813                         uint_t i;

815                         /* no alignment requirement for strings */
816                         for (i = 0; i < nelem; i++) {
817                                 if (strs[i] == NULL)
818                                         return (-1);
819                                 value_sz += strlen(strs[i]) + 1;
820                         }
821                 }
822                 break;
823         case DATA_TYPE_HRTIME:
824                 value_sz = sizeof (hrtime_t);
825                 break;
826         case DATA_TYPE_NVLIST:
827                 value_sz = NV_ALIGN(sizeof (nvlist_t));
828                 break;
829         case DATA_TYPE_NVLIST_ARRAY:
830                 value_sz = (uint64_t)nelem * sizeof (uint64_t) +
831                     (uint64_t)nelem * NV_ALIGN(sizeof (nvlist_t));
832                 break;
833         default:
834                 return (-1);
835         }

837         return (value_sz > INT32_MAX ? -1 : (int)value_sz);
838 }

840 static int
841 nvlist_copy_embedded(nvlist_t *nvl, nvlist_t *onvl, nvlist_t *emb_nvl)
842 {
843         nvpriv_t *priv;
844         int err;

846         if ((priv = nv_priv_alloc_embedded((nvpriv_t *)(uintptr_t)
847             nvl->nvl_priv)) == NULL)
848                 return (ENOMEM);

850         nvlist_init(emb_nvl, onvl->nvl_nvflag, priv);

852         if ((err = nvlist_copy_pairs(onvl, emb_nvl)) != 0) {
853                 nvlist_free(emb_nvl);
```



```
854                 emb_nvl->nvl_priv = 0;
855         }

857         return (err);
858 }

860 /*
861  * nvlist_add_common - Add new <name,value> pair to nvlist
862  */
863 static int
864 nvlist_add_common(nvlist_t *nvl, const char *name,
865     data_type_t type, uint_t nelem, const void *data)
866 {
867         nvpair_t *nvp;
868         uint_t i;

870         int nvp_sz, name_sz, value_sz;
871         int err = 0;

873         if (name == NULL || nvl == NULL || nvl->nvl_priv == 0)
874                 return (EINVAL);

876         if (nelem != 0 && data == NULL)
877                 return (EINVAL);

879         /*
880          * Verify type and nelem and get the value size.
881          * In case of data types DATA_TYPE_STRING and DATA_TYPE_STRING_ARRAY
882          * is the size of the string(s) included.
883          */
884         if ((value_sz = i_get_value_size(type, data, nelem)) < 0)
885                 return (EINVAL);

887         if (i_validate_nvpair_value(type, nelem, data) != 0)
888                 return (EINVAL);

890         /*
891          * If we're adding an nvlist or nvlist array, ensure that we are not
892          * adding the input nvlist to itself, which would cause recursion,
893          * and ensure that no NULL nvlist pointers are present.
894          */
895         switch (type) {
896         case DATA_TYPE_NVLIST:
897                 if (data == nvl || data == NULL)
898                         return (EINVAL);
899                 break;
900         case DATA_TYPE_NVLIST_ARRAY: {
901                 nvlist_t **onvlp = (nvlist_t **)data;
902                 for (i = 0; i < nelem; i++) {
903                         if (onvlp[i] == nvl || onvlp[i] == NULL)
904                                 return (EINVAL);
905                 }
906                 break;
907         }
908         default:
909                 break;
910         }

912         /* calculate sizes of the nvpair elements and the nvpair itself */
913         name_sz = strlen(name) + 1;

915         nvp_sz = NVP_SIZE_CALC(name_sz, value_sz);

917         if ((nvp = nvp_buf_alloc(nvl, nvp_sz)) == NULL)
918                 return (ENOMEM);
```

```
920         ASSERT(nvp->nvp_size == nvp_sz);
921         nvp->nvp_name_sz = name_sz;
922         nvp->nvp_value_elem = nelem;
923         nvp->nvp_type = type;
924         bcopy(name, NVP_NAME(nvp), name_sz);
925
926         switch (type) {
927         case DATA_TYPE_BOOLEAN:
928                 break;
929         case DATA_TYPE_STRING_ARRAY: {
930                 char *const *strs = data;
931                 char *buf = NVP_VALUE(nvp);
932                 char **cstrs = (void *)buf;
933
934                 /* skip pre-allocated space for pointer array */
935                 buf += nelem * sizeof (uint64_t);
936                 for (i = 0; i < nelem; i++) {
937                         int slen = strlen(strs[i]) + 1;
938                         bcopy(strs[i], buf, slen);
939                         cstrs[i] = buf;
940                         buf += slen;
941                 }
942                 break;
943         }
944         case DATA_TYPE_NVLIST: {
945                 nvlist_t *nnvl = EMBEDDED_NVL(nvp);
946                 nvlist_t *onvl = (nvlist_t *)data;
947
948                 if ((err = nvlist_copy_embedded(nvl, onvl, nnvl)) != 0) {
949                         nvp_buf_free(nvl, nvp);
950                         return (err);
951                 }
952                 break;
953         }
954         case DATA_TYPE_NVLIST_ARRAY: {
955                 nvlist_t **onvlp = (nvlist_t **)data;
956                 nvlist_t **nvlp = EMBEDDED_NVL_ARRAY(nvp);
957                 nvlist_t *embedded = (nvlist_t *)
958                     ((uintptr_t)nvlp + nelem * sizeof (uint64_t));
959
960                 for (i = 0; i < nelem; i++) {
961                         if ((err = nvlist_copy_embedded(nvl,
962                             onvlp[i], embedded)) != 0) {
963                                 /*
964                                  * Free any successfully created lists
965                                  */
966                                 nvpair_free(nvp);
967                                 nvp_buf_free(nvl, nvp);
968                                 return (err);
969                         }
970
971                         nvlp[i] = embedded++;
972                 }
973                 break;
974         }
975         default:
976                 bcopy(data, NVP_VALUE(nvp), value_sz);
977         }
978
979         /* if unique name, remove before add */
980         if (nvl->nvl_nvflag & NV_UNIQUE_NAME)
981                 (void) nvlist_remove_all(nvl, name);
982         else if (nvl->nvl_nvflag & NV_UNIQUE_NAME_TYPE)
983                 (void) nvlist_remove(nvl, name, type);
984
985         nvp_buf_link(nvl, nvp);
```



```
986
987         return (0);
988 }
989
990 int
991 nvlist_add_boolean(nvlist_t *nvl, const char *name)
992 {
993         return (nvlist_add_common(nvl, name, DATA_TYPE_BOOLEAN, 0, NULL));
994 }
995
996 int
997 nvlist_add_boolean_value(nvlist_t *nvl, const char *name, boolean_t val)
998 {
999         return (nvlist_add_common(nvl, name, DATA_TYPE_BOOLEAN_VALUE, 1, &val));
1000 }
1001
1002 int
1003 nvlist_add_byte(nvlist_t *nvl, const char *name, uchar_t val)
1004 {
1005         return (nvlist_add_common(nvl, name, DATA_TYPE_BYTE, 1, &val));
1006 }
1007
1008 int
1009 nvlist_add_int8(nvlist_t *nvl, const char *name, int8_t val)
1010 {
1011         return (nvlist_add_common(nvl, name, DATA_TYPE_INT8, 1, &val));
1012 }
1013
1014 int
1015 nvlist_add_uint8(nvlist_t *nvl, const char *name, uint8_t val)
1016 {
1017         return (nvlist_add_common(nvl, name, DATA_TYPE_UINT8, 1, &val));
1018 }
1019
1020 int
1021 nvlist_add_int16(nvlist_t *nvl, const char *name, int16_t val)
1022 {
1023         return (nvlist_add_common(nvl, name, DATA_TYPE_INT16, 1, &val));
1024 }
1025
1026 int
1027 nvlist_add_uint16(nvlist_t *nvl, const char *name, uint16_t val)
1028 {
1029         return (nvlist_add_common(nvl, name, DATA_TYPE_UINT16, 1, &val));
1030 }
1031
1032 int
1033 nvlist_add_int32(nvlist_t *nvl, const char *name, int32_t val)
1034 {
1035         return (nvlist_add_common(nvl, name, DATA_TYPE_INT32, 1, &val));
1036 }
1037
1038 int
1039 nvlist_add_uint32(nvlist_t *nvl, const char *name, uint32_t val)
1040 {
1041         return (nvlist_add_common(nvl, name, DATA_TYPE_UINT32, 1, &val));
1042 }
1043
1044 int
1045 nvlist_add_int64(nvlist_t *nvl, const char *name, int64_t val)
1046 {
1047         return (nvlist_add_common(nvl, name, DATA_TYPE_INT64, 1, &val));
1048 }
1049
1050 int
1051 nvlist_add_uint64(nvlist_t *nvl, const char *name, uint64_t val)
```

```
1052 {
1053         return (nvlist_add_common(nvl, name, DATA_TYPE_UINT64, 1, &val));
1054 }

1056 #if !defined(_KERNEL)
1057 int
1058 nvlist_add_double(nvlist_t *nvl, const char *name, double val)
1059 {
1060         return (nvlist_add_common(nvl, name, DATA_TYPE_DOUBLE, 1, &val));
1061 }
1062 #endif

1064 int
1065 nvlist_add_string(nvlist_t *nvl, const char *name, const char *val)
1066 {
1067         return (nvlist_add_common(nvl, name, DATA_TYPE_STRING, 1, (void *)val));
1068 }

1070 int
1071 nvlist_add_boolean_array(nvlist_t *nvl, const char *name,
1072     boolean_t *a, uint_t n)
1073 {
1074         return (nvlist_add_common(nvl, name, DATA_TYPE_BOOLEAN_ARRAY, n, a));
1075 }

1077 int
1078 nvlist_add_byte_array(nvlist_t *nvl, const char *name, uchar_t *a, uint_t n)
1079 {
1080         return (nvlist_add_common(nvl, name, DATA_TYPE_BYTE_ARRAY, n, a));
1081 }

1083 int
1084 nvlist_add_int8_array(nvlist_t *nvl, const char *name, int8_t *a, uint_t n)
1085 {
1086         return (nvlist_add_common(nvl, name, DATA_TYPE_INT8_ARRAY, n, a));
1087 }

1089 int
1090 nvlist_add_uint8_array(nvlist_t *nvl, const char *name, uint8_t *a, uint_t n)
1091 {
1092         return (nvlist_add_common(nvl, name, DATA_TYPE_UINT8_ARRAY, n, a));
1093 }

1095 int
1096 nvlist_add_int16_array(nvlist_t *nvl, const char *name, int16_t *a, uint_t n)
1097 {
1098         return (nvlist_add_common(nvl, name, DATA_TYPE_INT16_ARRAY, n, a));
1099 }

1101 int
1102 nvlist_add_uint16_array(nvlist_t *nvl, const char *name, uint16_t *a, uint_t n)
1103 {
1104         return (nvlist_add_common(nvl, name, DATA_TYPE_UINT16_ARRAY, n, a));
1105 }

1107 int
1108 nvlist_add_int32_array(nvlist_t *nvl, const char *name, int32_t *a, uint_t n)
1109 {
1110         return (nvlist_add_common(nvl, name, DATA_TYPE_INT32_ARRAY, n, a));
1111 }

1113 int
1114 nvlist_add_uint32_array(nvlist_t *nvl, const char *name, uint32_t *a, uint_t n)
1115 {
1116         return (nvlist_add_common(nvl, name, DATA_TYPE_UINT32_ARRAY, n, a));
1117 }
```



```
1119 int
1120 nvlist_add_int64_array(nvlist_t *nvl, const char *name, int64_t *a, uint_t n)
1121 {
1122         return (nvlist_add_common(nvl, name, DATA_TYPE_INT64_ARRAY, n, a));
1123 }

1125 int
1126 nvlist_add_uint64_array(nvlist_t *nvl, const char *name, uint64_t *a, uint_t n)
1127 {
1128         return (nvlist_add_common(nvl, name, DATA_TYPE_UINT64_ARRAY, n, a));
1129 }

1131 int
1132 nvlist_add_string_array(nvlist_t *nvl, const char *name,
1133     char *const *a, uint_t n)
1134 {
1135         return (nvlist_add_common(nvl, name, DATA_TYPE_STRING_ARRAY, n, a));
1136 }

1138 int
1139 nvlist_add_hrtime(nvlist_t *nvl, const char *name, hrtime_t val)
1140 {
1141         return (nvlist_add_common(nvl, name, DATA_TYPE_HRTIME, 1, &val));
1142 }

1144 int
1145 nvlist_add_nvlist(nvlist_t *nvl, const char *name, nvlist_t *val)
1146 {
1147         return (nvlist_add_common(nvl, name, DATA_TYPE_NVLIST, 1, val));
1148 }

1150 int
1151 nvlist_add_nvlist_array(nvlist_t *nvl, const char *name, nvlist_t **a, uint_t n)
1152 {
1153         return (nvlist_add_common(nvl, name, DATA_TYPE_NVLIST_ARRAY, n, a));
1154 }

1156 /* reading name-value pairs */
1157 nvpair_t *
1158 nvlist_next_nvpair(nvlist_t *nvl, nvpair_t *nvp)
1159 {
1160         nvpriv_t *priv;
1161         i_nvp_t *curr;

1163         if (nvl == NULL ||
1164             (priv = (nvpriv_t *)(uintptr_t)nvl->nvl_priv) == NULL)
1165                 return (NULL);

1167         curr = NVPAIR2I_NVP(nvp);

1169         /*
1170          * Ensure that nvp is a valid nvpair on this nvlist.
1171          * NB: nvp_curr is used only as a hint so that we don't always
1172          * have to walk the list to determine if nvp is still on the list.
1173          */
1174         if (nvp == NULL)
1175                 curr = priv->nvp_list;
1176         else if (priv->nvp_curr == curr || nvlist_contains_nvp(nvl, nvp))
1177                 curr = curr->nvi_next;
1178         else
1179                 curr = NULL;

1181         priv->nvp_curr = curr;

1183         return (curr != NULL ? &curr->nvi_nvp : NULL);
```

```
}

nvpair_t *
nvlist_prev_nvpair(nvlist_t *nvl, nvpair_t *nvp)
{
        nvpriv_t *priv;
        i_nvp_t *curr;

        if (nvl == NULL ||
            (priv = (nvpriv_t *)(uintptr_t)nvl->nvl_priv) == NULL)
                return (NULL);

        curr = NVPAIR2I_NVP(nvp);

        if (nvp == NULL)
                curr = priv->nvp_last;
        else if (priv->nvp_curr == curr || nvlist_contains_nvp(nvl, nvp))
                curr = curr->nvi_prev;
        else
                curr = NULL;

        priv->nvp_curr = curr;

        return (curr != NULL ? &curr->nvi_nvp : NULL);
}

boolean_t
nvlist_empty(nvlist_t *nvl)
{
        nvpriv_t *priv;

        if (nvl == NULL ||
            (priv = (nvpriv_t *)(uintptr_t)nvl->nvl_priv) == NULL)
                return (B_TRUE);

        return (priv->nvp_list == NULL);
}

char *
nvpair_name(nvpair_t *nvp)
{
        return (NVP_NAME(nvp));
}

data_type_t
nvpair_type(nvpair_t *nvp)
{
        return (NVP_TYPE(nvp));
}

int
nvpair_type_is_array(nvpair_t *nvp)
{
        data_type_t type = NVP_TYPE(nvp);

        if ((type == DATA_TYPE_BYTE_ARRAY) ||
            (type == DATA_TYPE_UINT8_ARRAY) ||
            (type == DATA_TYPE_INT16_ARRAY) ||
            (type == DATA_TYPE_UINT16_ARRAY) ||
            (type == DATA_TYPE_INT32_ARRAY) ||
            (type == DATA_TYPE_UINT32_ARRAY) ||
            (type == DATA_TYPE_INT64_ARRAY) ||
            (type == DATA_TYPE_UINT64_ARRAY) ||
            (type == DATA_TYPE_BOOLEAN_ARRAY) ||
            (type == DATA_TYPE_STRING_ARRAY) ||
            (type == DATA_TYPE_NVLIST_ARRAY))
```



```
                return (1);
        return (0);

}

static int
nvpair_value_common(nvpair_t *nvp, data_type_t type, uint_t *nelem, void *data)
{
        if (nvp == NULL || nvpair_type(nvp) != type)
                return (EINVAL);

        /*
         * For non-array types, we copy the data.
         * For array types (including string), we set a pointer.
         */
        switch (type) {
        case DATA_TYPE_BOOLEAN:
                if (nelem != NULL)
                        *nelem = 0;
                break;

        case DATA_TYPE_BOOLEAN_VALUE:
        case DATA_TYPE_BYTE:
        case DATA_TYPE_INT8:
        case DATA_TYPE_UINT8:
        case DATA_TYPE_INT16:
        case DATA_TYPE_UINT16:
        case DATA_TYPE_INT32:
        case DATA_TYPE_UINT32:
        case DATA_TYPE_INT64:
        case DATA_TYPE_UINT64:
        case DATA_TYPE_HRTIME:
#if !defined(_KERNEL)
        case DATA_TYPE_DOUBLE:
#endif
                if (data == NULL)
                        return (EINVAL);
                bcopy(NVP_VALUE(nvp), data,
                    (size_t)i_get_value_size(type, NULL, 1));
                if (nelem != NULL)
                        *nelem = 1;
                break;

        case DATA_TYPE_NVLIST:
        case DATA_TYPE_STRING:
                if (data == NULL)
                        return (EINVAL);
                *(void **)data = (void *)NVP_VALUE(nvp);
                if (nelem != NULL)
                        *nelem = 1;
                break;

        case DATA_TYPE_BOOLEAN_ARRAY:
        case DATA_TYPE_BYTE_ARRAY:
        case DATA_TYPE_INT8_ARRAY:
        case DATA_TYPE_UINT8_ARRAY:
        case DATA_TYPE_INT16_ARRAY:
        case DATA_TYPE_UINT16_ARRAY:
        case DATA_TYPE_INT32_ARRAY:
        case DATA_TYPE_UINT32_ARRAY:
        case DATA_TYPE_INT64_ARRAY:
        case DATA_TYPE_UINT64_ARRAY:
        case DATA_TYPE_STRING_ARRAY:
        case DATA_TYPE_NVLIST_ARRAY:
                if (nelem == NULL || data == NULL)
                        return (EINVAL);
```

```
1316                 if ((*nelem = NVP_NELEM(nvp)) != 0)
1317                         *(void **)data = (void *)NVP_VALUE(nvp);
1318                 else
1319                         *(void **)data = NULL;
1320                 break;
1322         default:
1323                 return (ENOTSUP);
1324         }
1326         return (0);
1327 }
1329 static int
1330 nvlist_lookup_common(nvlist_t *nvl, const char *name, data_type_t type,
1331     uint_t *nelem, void *data)
1332 {
1333         nvpriv_t *priv;
1334         nvpair_t *nvp;
1335         i_nvp_t *curr;
1337         if (name == NULL || nvl == NULL ||
1338             (priv = (nvpriv_t *)(uintptr_t)nvl->nvl_priv) == NULL)
1339                 return (EINVAL);
1341         if (!(nvl->nvl_nvflag & (NV_UNIQUE_NAME | NV_UNIQUE_NAME_TYPE)))
1342                 return (ENOTSUP);
1344         for (curr = priv->nvp_list; curr != NULL; curr = curr->nvi_next) {
1345                 nvp = &curr->nvi_nvp;
1347                 if (strcmp(name, NVP_NAME(nvp)) == 0 && NVP_TYPE(nvp) == type)
1348                         return (nvpair_value_common(nvp, type, nelem, data));
1349         }
1351         return (ENOENT);
1352 }
1354 int
1355 nvlist_lookup_boolean(nvlist_t *nvl, const char *name)
1356 {
1357         return (nvlist_lookup_common(nvl, name, DATA_TYPE_BOOLEAN, NULL, NULL));
1358 }
1360 int
1361 nvlist_lookup_boolean_value(nvlist_t *nvl, const char *name, boolean_t *val)
1362 {
1363         return (nvlist_lookup_common(nvl, name,
1364             DATA_TYPE_BOOLEAN_VALUE, NULL, val));
1365 }
1367 int
1368 nvlist_lookup_byte(nvlist_t *nvl, const char *name, uchar_t *val)
1369 {
1370         return (nvlist_lookup_common(nvl, name, DATA_TYPE_BYTE, NULL, val));
1371 }
1373 int
1374 nvlist_lookup_int8(nvlist_t *nvl, const char *name, int8_t *val)
1375 {
1376         return (nvlist_lookup_common(nvl, name, DATA_TYPE_INT8, NULL, val));
1377 }
1379 int
1380 nvlist_lookup_uint8(nvlist_t *nvl, const char *name, uint8_t *val)
1381 {
```



```
1382         return (nvlist_lookup_common(nvl, name, DATA_TYPE_UINT8, NULL, val));
1383 }
1385 int
1386 nvlist_lookup_int16(nvlist_t *nvl, const char *name, int16_t *val)
1387 {
1388         return (nvlist_lookup_common(nvl, name, DATA_TYPE_INT16, NULL, val));
1389 }
1391 int
1392 nvlist_lookup_uint16(nvlist_t *nvl, const char *name, uint16_t *val)
1393 {
1394         return (nvlist_lookup_common(nvl, name, DATA_TYPE_UINT16, NULL, val));
1395 }
1397 int
1398 nvlist_lookup_int32(nvlist_t *nvl, const char *name, int32_t *val)
1399 {
1400         return (nvlist_lookup_common(nvl, name, DATA_TYPE_INT32, NULL, val));
1401 }
1403 int
1404 nvlist_lookup_uint32(nvlist_t *nvl, const char *name, uint32_t *val)
1405 {
1406         return (nvlist_lookup_common(nvl, name, DATA_TYPE_UINT32, NULL, val));
1407 }
1409 int
1410 nvlist_lookup_int64(nvlist_t *nvl, const char *name, int64_t *val)
1411 {
1412         return (nvlist_lookup_common(nvl, name, DATA_TYPE_INT64, NULL, val));
1413 }
1415 int
1416 nvlist_lookup_uint64(nvlist_t *nvl, const char *name, uint64_t *val)
1417 {
1418         return (nvlist_lookup_common(nvl, name, DATA_TYPE_UINT64, NULL, val));
1419 }
1421 #if !defined(_KERNEL)
1422 int
1423 nvlist_lookup_double(nvlist_t *nvl, const char *name, double *val)
1424 {
1425         return (nvlist_lookup_common(nvl, name, DATA_TYPE_DOUBLE, NULL, val));
1426 }
1427 #endif
1429 int
1430 nvlist_lookup_string(nvlist_t *nvl, const char *name, char **val)
1431 {
1432         return (nvlist_lookup_common(nvl, name, DATA_TYPE_STRING, NULL, val));
1433 }
1435 int
1436 nvlist_lookup_nvlist(nvlist_t *nvl, const char *name, nvlist_t **val)
1437 {
1438         return (nvlist_lookup_common(nvl, name, DATA_TYPE_NVLIST, NULL, val));
1439 }
1441 int
1442 nvlist_lookup_boolean_array(nvlist_t *nvl, const char *name,
1443     boolean_t **a, uint_t *n)
1444 {
1445         return (nvlist_lookup_common(nvl, name,
1446             DATA_TYPE_BOOLEAN_ARRAY, n, a));
1447 }
```

```
1449 int
1450 nvlist_lookup_byte_array(nvlist_t *nvl, const char *name,
1451     uchar_t **a, uint_t *n)
1452 {
1453         return (nvlist_lookup_common(nvl, name, DATA_TYPE_BYTE_ARRAY, n, a));
1454 }

1456 int
1457 nvlist_lookup_int8_array(nvlist_t *nvl, const char *name, int8_t **a, uint_t *n)
1458 {
1459         return (nvlist_lookup_common(nvl, name, DATA_TYPE_INT8_ARRAY, n, a));
1460 }

1462 int
1463 nvlist_lookup_uint8_array(nvlist_t *nvl, const char *name,
1464     uint8_t **a, uint_t *n)
1465 {
1466         return (nvlist_lookup_common(nvl, name, DATA_TYPE_UINT8_ARRAY, n, a));
1467 }

1469 int
1470 nvlist_lookup_int16_array(nvlist_t *nvl, const char *name,
1471     int16_t **a, uint_t *n)
1472 {
1473         return (nvlist_lookup_common(nvl, name, DATA_TYPE_INT16_ARRAY, n, a));
1474 }

1476 int
1477 nvlist_lookup_uint16_array(nvlist_t *nvl, const char *name,
1478     uint16_t **a, uint_t *n)
1479 {
1480         return (nvlist_lookup_common(nvl, name, DATA_TYPE_UINT16_ARRAY, n, a));
1481 }

1483 int
1484 nvlist_lookup_int32_array(nvlist_t *nvl, const char *name,
1485     int32_t **a, uint_t *n)
1486 {
1487         return (nvlist_lookup_common(nvl, name, DATA_TYPE_INT32_ARRAY, n, a));
1488 }

1490 int
1491 nvlist_lookup_uint32_array(nvlist_t *nvl, const char *name,
1492     uint32_t **a, uint_t *n)
1493 {
1494         return (nvlist_lookup_common(nvl, name, DATA_TYPE_UINT32_ARRAY, n, a));
1495 }

1497 int
1498 nvlist_lookup_int64_array(nvlist_t *nvl, const char *name,
1499     int64_t **a, uint_t *n)
1500 {
1501         return (nvlist_lookup_common(nvl, name, DATA_TYPE_INT64_ARRAY, n, a));
1502 }

1504 int
1505 nvlist_lookup_uint64_array(nvlist_t *nvl, const char *name,
1506     uint64_t **a, uint_t *n)
1507 {
1508         return (nvlist_lookup_common(nvl, name, DATA_TYPE_UINT64_ARRAY, n, a));
1509 }

1511 int
1512 nvlist_lookup_string_array(nvlist_t *nvl, const char *name,
1513     char ***a, uint_t *n)
```



```
1514 {
1515         return (nvlist_lookup_common(nvl, name, DATA_TYPE_STRING_ARRAY, n, a));
1516 }

1518 int
1519 nvlist_lookup_nvlist_array(nvlist_t *nvl, const char *name,
1520     nvlist_t ***a, uint_t *n)
1521 {
1522         return (nvlist_lookup_common(nvl, name, DATA_TYPE_NVLIST_ARRAY, n, a));
1523 }

1525 int
1526 nvlist_lookup_hrtime(nvlist_t *nvl, const char *name, hrtime_t *val)
1527 {
1528         return (nvlist_lookup_common(nvl, name, DATA_TYPE_HRTIME, NULL, val));
1529 }

1531 int
1532 nvlist_lookup_pairs(nvlist_t *nvl, int flag, ...)
1533 {
1534         va_list ap;
1535         char *name;
1536         int noentok = (flag & NV_FLAG_NOENTOK ? 1 : 0);
1537         int ret = 0;

1539         va_start(ap, flag);
1540         while (ret == 0 && (name = va_arg(ap, char *)) != NULL) {
1541                 data_type_t type;
1542                 void *val;
1543                 uint_t *nelem;

1545                 switch (type = va_arg(ap, data_type_t)) {
1546                 case DATA_TYPE_BOOLEAN:
1547                         ret = nvlist_lookup_common(nvl, name, type, NULL, NULL);
1548                         break;

1550                 case DATA_TYPE_BOOLEAN_VALUE:
1551                 case DATA_TYPE_BYTE:
1552                 case DATA_TYPE_INT8:
1553                 case DATA_TYPE_UINT8:
1554                 case DATA_TYPE_INT16:
1555                 case DATA_TYPE_UINT16:
1556                 case DATA_TYPE_INT32:
1557                 case DATA_TYPE_UINT32:
1558                 case DATA_TYPE_INT64:
1559                 case DATA_TYPE_UINT64:
1560                 case DATA_TYPE_HRTIME:
1561                 case DATA_TYPE_STRING:
1562                 case DATA_TYPE_NVLIST:
1563 #if !defined(_KERNEL)
1564                 case DATA_TYPE_DOUBLE:
1565 #endif
1566                         val = va_arg(ap, void *);
1567                         ret = nvlist_lookup_common(nvl, name, type, NULL, val);
1568                         break;

1570                 case DATA_TYPE_BYTE_ARRAY:
1571                 case DATA_TYPE_BOOLEAN_ARRAY:
1572                 case DATA_TYPE_INT8_ARRAY:
1573                 case DATA_TYPE_UINT8_ARRAY:
1574                 case DATA_TYPE_INT16_ARRAY:
1575                 case DATA_TYPE_UINT16_ARRAY:
1576                 case DATA_TYPE_INT32_ARRAY:
1577                 case DATA_TYPE_UINT32_ARRAY:
1578                 case DATA_TYPE_INT64_ARRAY:
1579                 case DATA_TYPE_UINT64_ARRAY:
```

```
1580                 case DATA_TYPE_STRING_ARRAY:
1581                 case DATA_TYPE_NVLIST_ARRAY:
1582                         val = va_arg(ap, void *);
1583                         nelem = va_arg(ap, uint_t *);
1584                         ret = nvlist_lookup_common(nvl, name, type, nelem, val);
1585                         break;

1587                 default:
1588                         ret = EINVAL;
1589                 }

1591                 if (ret == ENOENT && noentok)
1592                         ret = 0;
1593         }
1594         va_end(ap);

1596         return (ret);
1597 }

1599 /*
1600  * Find the 'name'ed nvpair in the nvlist 'nvl'. If 'name' found, the function
1601  * returns zero and a pointer to the matching nvpair is returned in '*ret'
1602  * (given 'ret' is non-NULL). If 'sep' is specified then 'name' will penitrate
1603  * multiple levels of embedded nvlists, with 'sep' as the separator. As an
1604  * example, if sep is '.', name might look like: "a" or "a.b" or "a.c[3]" or
1605  * "a.d[3].e[1]".  This matches the C syntax for array embed (for convience,
1606  * code also supports "a.d[3]e[1]" syntax).
1607  *
1608  * If 'ip' is non-NULL and the last name component is an array, return the
1609  * value of the "...[index]" array index in *ip. For an array reference that
1610  * is not indexed, *ip will be returned as -1. If there is a syntax error in
1611  * 'name', and 'ep' is non-NULL then *ep will be set to point to the location
1612  * inside the 'name' string where the syntax error was detected.
1613  */
1614 static int
1615 nvlist_lookup_nvpair_ei_sep(nvlist_t *nvl, const char *name, const char sep,
1616     nvpair_t **ret, int *ip, char **ep)
1617 {
1618         nvpair_t        *nvp;
1619         const char      *np;
1620         char            *sepp;
1621         char            *idxp, *idxep;
1622         nvlist_t        **nva;
1623         long            idx;
1624         int             n;

1626         if (ip)
1627                 *ip = -1;                       /* not indexed */
1628         if (ep)
1629                 *ep = NULL;

1631         if ((nvl == NULL) || (name == NULL))
1632                 return (EINVAL);

1634         /* step through components of name */
1635         for (np = name; np && *np; np = sepp) {
1636                 /* ensure unique names */
1637                 if (!(nvl->nvl_nvflag & NV_UNIQUE_NAME))
1638                         return (ENOTSUP);

1640                 /* skip white space */
1641                 skip_whitespace(np);
1642                 if (*np == 0)
1643                         break;

1645                 /* set 'sepp' to end of current component 'np' */
```



```
1646                 if (sep)
1647                         sepp = strchr(np, sep);
1648                 else
1649                         sepp = NULL;

1651                 /* find start of next "[ index ]..." */
1652                 idxp = strchr(np, '[');

1654                 /* if sepp comes first, set idxp to NULL */
1655                 if (sepp && idxp && (sepp < idxp))
1656                         idxp = NULL;

1658                 /*
1659                  * At this point 'idxp' is set if there is an index
1660                  * expected for the current component.
1661                  */
1662                 if (idxp) {
1663                         /* set 'n' to length of current 'np' name component */
1664                         n = idxp++ - np;

1666                         /* keep sepp up to date for *ep use as we advance */
1667                         skip_whitespace(idxp);
1668                         sepp = idxp;

1670                         /* determine the index value */
1671 #if defined(_KERNEL) && !defined(_BOOT)
1672                         if (ddi_strtol(idxp, &idxep, 0, &idx))
1673                                 goto fail;
1674 #else
1675                         idx = strtol(idxp, &idxep, 0);
1676 #endif
1677                         if (idxep == idxp)
1678                                 goto fail;

1680                         /* keep sepp up to date for *ep use as we advance */
1681                         sepp = idxep;

1683                         /* skip white space index value and check for ']' */
1684                         skip_whitespace(sepp);
1685                         if (*sepp++ != ']')
1686                                 goto fail;

1688                         /* for embedded arrays, support C syntax: "a[1].b" */
1689                         skip_whitespace(sepp);
1690                         if (sep && (*sepp == sep))
1691                                 sepp++;
1692                 } else if (sepp) {
1693                         n = sepp++ - np;
1694                 } else {
1695                         n = strlen(np);
1696                 }

1698                 /* trim trailing whitespace by reducing length of 'np' */
1699                 if (n == 0)
1700                         goto fail;
1701                 for (n--; (np[n] == ' ') || (np[n] == '\t'); n--)
1702                         ;
1703                 n++;

1705                 /* skip whitespace, and set sepp to NULL if complete */
1706                 if (sepp) {
1707                         skip_whitespace(sepp);
1708                         if (*sepp == 0)
1709                                 sepp = NULL;
1710                 }
```

```
1712                 /*
1713                  * At this point:
1714                  * o  'n' is the length of current 'np' component.
1715                  * o  'idxp' is set if there was an index, and value 'idx'.
1716                  * o  'sepp' is set to the beginning of the next component,
1717                  *    and set to NULL if we have no more components.
1718                  *
1719                  * Search for nvpair with matching component name.
1720                  */
1721                 for (nvp = nvlist_next_nvpair(nvl, NULL); nvp != NULL;
1722                     nvp = nvlist_next_nvpair(nvl, nvp)) {

1724                         /* continue if no match on name */
1725                         if (strncmp(np, nvpair_name(nvp), n) ||
1726                             (strlen(nvpair_name(nvp)) != n))
1727                                 continue;

1729                         /* if indexed, verify type is array oriented */
1730                         if (idxp && !nvpair_type_is_array(nvp))
1731                                 goto fail;

1733                         /*
1734                          * Full match found, return nvp and idx if this
1735                          * was the last component.
1736                          */
1737                         if (sepp == NULL) {
1738                                 if (ret)
1739                                         *ret = nvp;
1740                                 if (ip && idxp)
1741                                         *ip = (int)idx; /* return index */
1742                                 return (0);             /* found */
1743                         }

1745                         /*
1746                          * More components: current match must be
1747                          * of DATA_TYPE_NVLIST or DATA_TYPE_NVLIST_ARRAY
1748                          * to support going deeper.
1749                          */
1750                         if (nvpair_type(nvp) == DATA_TYPE_NVLIST) {
1751                                 nvl = EMBEDDED_NVL(nvp);
1752                                 break;
1753                         } else if (nvpair_type(nvp) == DATA_TYPE_NVLIST_ARRAY) {
1754                                 (void) nvpair_value_nvlist_array(nvp,
1755                                     &nva, (uint_t *)&n);
1756                                 if ((n < 0) || (idx >= n))
1757                                         goto fail;
1758                                 nvl = nva[idx];
1759                                 break;
1760                         }

1762                         /* type does not support more levels */
1763                         goto fail;
1764                 }
1765                 if (nvp == NULL)
1766                         goto fail;              /* 'name' not found */

1768                 /* search for match of next component in embedded 'nvl' list */
1769         }

1771 fail:   if (ep && sepp)
1772                 *ep = sepp;
1773         return (EINVAL);
1774 }

1776 /*
1777  * Return pointer to nvpair with specified 'name'.
```



```
1778  */
1779 int
1780 nvlist_lookup_nvpair(nvlist_t *nvl, const char *name, nvpair_t **ret)
1781 {
1782         return (nvlist_lookup_nvpair_ei_sep(nvl, name, 0, ret, NULL, NULL));
1783 }

1785 /*
1786  * Determine if named nvpair exists in nvlist (use embedded separator of '.'
1787  * and return array index).  See nvlist_lookup_nvpair_ei_sep for more detailed
1788  * description.
1789  */
1790 int nvlist_lookup_nvpair_embedded_index(nvlist_t *nvl,
1791     const char *name, nvpair_t **ret, int *ip, char **ep)
1792 {
1793         return (nvlist_lookup_nvpair_ei_sep(nvl, name, '.', ret, ip, ep));
1794 }

1796 boolean_t
1797 nvlist_exists(nvlist_t *nvl, const char *name)
1798 {
1799         nvpriv_t *priv;
1800         nvpair_t *nvp;
1801         i_nvp_t *curr;

1803         if (name == NULL || nvl == NULL ||
1804             (priv = (nvpriv_t *)(uintptr_t)nvl->nvl_priv) == NULL)
1805                 return (B_FALSE);

1807         for (curr = priv->nvp_list; curr != NULL; curr = curr->nvi_next) {
1808                 nvp = &curr->nvi_nvp;

1810                 if (strcmp(name, NVP_NAME(nvp)) == 0)
1811                         return (B_TRUE);
1812         }

1814         return (B_FALSE);
1815 }

1817 int
1818 nvpair_value_boolean_value(nvpair_t *nvp, boolean_t *val)
1819 {
1820         return (nvpair_value_common(nvp, DATA_TYPE_BOOLEAN_VALUE, NULL, val));
1821 }

1823 int
1824 nvpair_value_byte(nvpair_t *nvp, uchar_t *val)
1825 {
1826         return (nvpair_value_common(nvp, DATA_TYPE_BYTE, NULL, val));
1827 }

1829 int
1830 nvpair_value_int8(nvpair_t *nvp, int8_t *val)
1831 {
1832         return (nvpair_value_common(nvp, DATA_TYPE_INT8, NULL, val));
1833 }

1835 int
1836 nvpair_value_uint8(nvpair_t *nvp, uint8_t *val)
1837 {
1838         return (nvpair_value_common(nvp, DATA_TYPE_UINT8, NULL, val));
1839 }

1841 int
1842 nvpair_value_int16(nvpair_t *nvp, int16_t *val)
1843 {
```

```
1844         return (nvpair_value_common(nvp, DATA_TYPE_INT16, NULL, val));
1845 }

1847 int
1848 nvpair_value_uint16(nvpair_t *nvp, uint16_t *val)
1849 {
1850         return (nvpair_value_common(nvp, DATA_TYPE_UINT16, NULL, val));
1851 }

1853 int
1854 nvpair_value_int32(nvpair_t *nvp, int32_t *val)
1855 {
1856         return (nvpair_value_common(nvp, DATA_TYPE_INT32, NULL, val));
1857 }

1859 int
1860 nvpair_value_uint32(nvpair_t *nvp, uint32_t *val)
1861 {
1862         return (nvpair_value_common(nvp, DATA_TYPE_UINT32, NULL, val));
1863 }

1865 int
1866 nvpair_value_int64(nvpair_t *nvp, int64_t *val)
1867 {
1868         return (nvpair_value_common(nvp, DATA_TYPE_INT64, NULL, val));
1869 }

1871 int
1872 nvpair_value_uint64(nvpair_t *nvp, uint64_t *val)
1873 {
1874         return (nvpair_value_common(nvp, DATA_TYPE_UINT64, NULL, val));
1875 }

1877 #if !defined(_KERNEL)
1878 int
1879 nvpair_value_double(nvpair_t *nvp, double *val)
1880 {
1881         return (nvpair_value_common(nvp, DATA_TYPE_DOUBLE, NULL, val));
1882 }
1883 #endif

1885 int
1886 nvpair_value_string(nvpair_t *nvp, char **val)
1887 {
1888         return (nvpair_value_common(nvp, DATA_TYPE_STRING, NULL, val));
1889 }

1891 int
1892 nvpair_value_nvlist(nvpair_t *nvp, nvlist_t **val)
1893 {
1894         return (nvpair_value_common(nvp, DATA_TYPE_NVLIST, NULL, val));
1895 }

1897 int
1898 nvpair_value_boolean_array(nvpair_t *nvp, boolean_t **val, uint_t *nelem)
1899 {
1900         return (nvpair_value_common(nvp, DATA_TYPE_BOOLEAN_ARRAY, nelem, val));
1901 }

1903 int
1904 nvpair_value_byte_array(nvpair_t *nvp, uchar_t **val, uint_t *nelem)
1905 {
1906         return (nvpair_value_common(nvp, DATA_TYPE_BYTE_ARRAY, nelem, val));
1907 }

1909 int
```



```
1910 nvpair_value_int8_array(nvpair_t *nvp, int8_t **val, uint_t *nelem)
1911 {
1912         return (nvpair_value_common(nvp, DATA_TYPE_INT8_ARRAY, nelem, val));
1913 }

1915 int
1916 nvpair_value_uint8_array(nvpair_t *nvp, uint8_t **val, uint_t *nelem)
1917 {
1918         return (nvpair_value_common(nvp, DATA_TYPE_UINT8_ARRAY, nelem, val));
1919 }

1921 int
1922 nvpair_value_int16_array(nvpair_t *nvp, int16_t **val, uint_t *nelem)
1923 {
1924         return (nvpair_value_common(nvp, DATA_TYPE_INT16_ARRAY, nelem, val));
1925 }

1927 int
1928 nvpair_value_uint16_array(nvpair_t *nvp, uint16_t **val, uint_t *nelem)
1929 {
1930         return (nvpair_value_common(nvp, DATA_TYPE_UINT16_ARRAY, nelem, val));
1931 }

1933 int
1934 nvpair_value_int32_array(nvpair_t *nvp, int32_t **val, uint_t *nelem)
1935 {
1936         return (nvpair_value_common(nvp, DATA_TYPE_INT32_ARRAY, nelem, val));
1937 }

1939 int
1940 nvpair_value_uint32_array(nvpair_t *nvp, uint32_t **val, uint_t *nelem)
1941 {
1942         return (nvpair_value_common(nvp, DATA_TYPE_UINT32_ARRAY, nelem, val));
1943 }

1945 int
1946 nvpair_value_int64_array(nvpair_t *nvp, int64_t **val, uint_t *nelem)
1947 {
1948         return (nvpair_value_common(nvp, DATA_TYPE_INT64_ARRAY, nelem, val));
1949 }

1951 int
1952 nvpair_value_uint64_array(nvpair_t *nvp, uint64_t **val, uint_t *nelem)
1953 {
1954         return (nvpair_value_common(nvp, DATA_TYPE_UINT64_ARRAY, nelem, val));
1955 }

1957 int
1958 nvpair_value_string_array(nvpair_t *nvp, char ***val, uint_t *nelem)
1959 {
1960         return (nvpair_value_common(nvp, DATA_TYPE_STRING_ARRAY, nelem, val));
1961 }

1963 int
1964 nvpair_value_nvlist_array(nvpair_t *nvp, nvlist_t ***val, uint_t *nelem)
1965 {
1966         return (nvpair_value_common(nvp, DATA_TYPE_NVLIST_ARRAY, nelem, val));
1967 }

1969 int
1970 nvpair_value_hrtime(nvpair_t *nvp, hrtime_t *val)
1971 {
1972         return (nvpair_value_common(nvp, DATA_TYPE_HRTIME, NULL, val));
1973 }

1975 /*
```

```
1976  * Add specified pair to the list.
1977  */
1978 int
1979 nvlist_add_nvpair(nvlist_t *nvl, nvpair_t *nvp)
1980 {
1981         if (nvl == NULL || nvp == NULL)
1982                 return (EINVAL);

1984         return (nvlist_add_common(nvl, NVP_NAME(nvp), NVP_TYPE(nvp),
1985             NVP_NELEM(nvp), NVP_VALUE(nvp)));
1986 }

1988 /*
1989  * Merge the supplied nvlists and put the result in dst.
1990  * The merged list will contain all names specified in both lists,
1991  * the values are taken from nvl in the case of duplicates.
1992  * Return 0 on success.
1993  */
1994 /*ARGSUSED*/
1995 int
1996 nvlist_merge(nvlist_t *dst, nvlist_t *nvl, int flag)
1997 {
1998         if (nvl == NULL || dst == NULL)
1999                 return (EINVAL);

2001         if (dst != nvl)
2002                 return (nvlist_copy_pairs(nvl, dst));

2004         return (0);
2005 }

2007 /*
2008  * Encoding related routines
2009  */
2010 #define NVS_OP_ENCODE   0
2011 #define NVS_OP_DECODE   1
2012 #define NVS_OP_GETSIZE  2

2014 typedef struct nvs_ops nvs_ops_t;

2016 typedef struct {
2017         int             nvs_op;
2018         const nvs_ops_t *nvs_ops;
2019         void            *nvs_private;
2020         nvpriv_t        *nvs_priv;
2021 } nvstream_t;

2023 /*
2024  * nvs operations are:
2025  *   - nvs_nvlist
2026  *     encoding / decoding of a nvlist header (nvlist_t)
2027  *     calculates the size used for header and end detection
2028  *
2029  *   - nvs_nvpair
2030  *     responsible for the first part of encoding / decoding of an nvpair
2031  *     calculates the decoded size of an nvpair
2032  *
2033  *   - nvs_nvp_op
2034  *     second part of encoding / decoding of an nvpair
2035  *
2036  *   - nvs_nvp_size
2037  *     calculates the encoding size of an nvpair
2038  *
2039  *   - nvs_nvl_fini
2040  *     encodes the end detection mark (zeros).
2041  */
```



```
2042 struct nvs_ops {
2043         int (*nvs_nvlist)(nvstream_t *, nvlist_t *, size_t *);
2044         int (*nvs_nvpair)(nvstream_t *, nvpair_t *, size_t *);
2045         int (*nvs_nvp_op)(nvstream_t *, nvpair_t *);
2046         int (*nvs_nvp_size)(nvstream_t *, nvpair_t *, size_t *);
2047         int (*nvs_nvl_fini)(nvstream_t *);
2048 };

2050 typedef struct {
2051         char    nvh_encoding;   /* nvs encoding method */
2052         char    nvh_endian;     /* nvs endian */
2053         char    nvh_reserved1;  /* reserved for future use */
2054         char    nvh_reserved2;  /* reserved for future use */
2055 } nvs_header_t;

2057 static int
2058 nvs_encode_pairs(nvstream_t *nvs, nvlist_t *nvl)
2059 {
2060         nvpriv_t *priv = (nvpriv_t *)(uintptr_t)nvl->nvl_priv;
2061         i_nvp_t *curr;

2063         /*
2064          * Walk nvpair in list and encode each nvpair
2065          */
2066         for (curr = priv->nvp_list; curr != NULL; curr = curr->nvi_next)
2067                 if (nvs->nvs_ops->nvs_nvpair(nvs, &curr->nvi_nvp, NULL) != 0)
2068                         return (EFAULT);

2070         return (nvs->nvs_ops->nvs_nvl_fini(nvs));
2071 }

2073 static int
2074 nvs_decode_pairs(nvstream_t *nvs, nvlist_t *nvl)
2075 {
2076         nvpair_t *nvp;
2077         size_t nvsize;
2078         int err;

2080         /*
2081          * Get decoded size of next pair in stream, alloc
2082          * memory for nvpair_t, then decode the nvpair
2083          */
2084         while ((err = nvs->nvs_ops->nvs_nvpair(nvs, NULL, &nvsize)) == 0) {
2085                 if (nvsize == 0) /* end of list */
2086                         break;

2088                 /* make sure len makes sense */
2089                 if (nvsize < NVP_SIZE_CALC(1, 0))
2090                         return (EFAULT);

2092                 if ((nvp = nvp_buf_alloc(nvl, nvsize)) == NULL)
2093                         return (ENOMEM);

2095                 if ((err = nvs->nvs_ops->nvs_nvp_op(nvs, nvp)) != 0) {
2096                         nvp_buf_free(nvl, nvp);
2097                         return (err);
2098                 }

2100                 if (i_validate_nvpair(nvp) != 0) {
2101                         nvpair_free(nvp);
2102                         nvp_buf_free(nvl, nvp);
2103                         return (EFAULT);
2104                 }

2106                 nvp_buf_link(nvl, nvp);
2107         }
```

```
2108         return (err);
2109 }

2111 static int
2112 nvs_getsize_pairs(nvstream_t *nvs, nvlist_t *nvl, size_t *buflen)
2113 {
2114         nvpriv_t *priv = (nvpriv_t *)(uintptr_t)nvl->nvl_priv;
2115         i_nvp_t *curr;
2116         uint64_t nvsize = *buflen;
2117         size_t size;

2119         /*
2120          * Get encoded size of nvpairs in nvlist
2121          */
2122         for (curr = priv->nvp_list; curr != NULL; curr = curr->nvi_next) {
2123                 if (nvs->nvs_ops->nvs_nvp_size(nvs, &curr->nvi_nvp, &size) != 0)
2124                         return (EINVAL);

2126                 if ((nvsize += size) > INT32_MAX)
2127                         return (EINVAL);
2128         }

2130         *buflen = nvsize;
2131         return (0);
2132 }

2134 static int
2135 nvs_operation(nvstream_t *nvs, nvlist_t *nvl, size_t *buflen)
2136 {
2137         int err;

2139         if (nvl->nvl_priv == 0)
2140                 return (EFAULT);

2142         /*
2143          * Perform the operation, starting with header, then each nvpair
2144          */
2145         if ((err = nvs->nvs_ops->nvs_nvlist(nvs, nvl, buflen)) != 0)
2146                 return (err);

2148         switch (nvs->nvs_op) {
2149         case NVS_OP_ENCODE:
2150                 err = nvs_encode_pairs(nvs, nvl);
2151                 break;

2153         case NVS_OP_DECODE:
2154                 err = nvs_decode_pairs(nvs, nvl);
2155                 break;

2157         case NVS_OP_GETSIZE:
2158                 err = nvs_getsize_pairs(nvs, nvl, buflen);
2159                 break;

2161         default:
2162                 err = EINVAL;
2163         }

2165         return (err);
2166 }

2168 static int
2169 nvs_embedded(nvstream_t *nvs, nvlist_t *embedded)
2170 {
2171         switch (nvs->nvs_op) {
2172         case NVS_OP_ENCODE:
2173                 return (nvs_operation(nvs, embedded, NULL));
```



```
2175         case NVS_OP_DECODE: {
2176                 nvpriv_t *priv;
2177                 int err;

2179                 if (embedded->nvl_version != NV_VERSION)
2180                         return (ENOTSUP);

2182                 if ((priv = nv_priv_alloc_embedded(nvs->nvs_priv)) == NULL)
2183                         return (ENOMEM);

2185                 nvlist_init(embedded, embedded->nvl_nvflag, priv);

2187                 if ((err = nvs_operation(nvs, embedded, NULL)) != 0)
2188                         nvlist_free(embedded);
2189                 return (err);
2190         }
2191         default:
2192                 break;
2193         }

2195         return (EINVAL);
2196 }

2198 static int
2199 nvs_embedded_nvl_array(nvstream_t *nvs, nvpair_t *nvp, size_t *size)
2200 {
2201         size_t nelem = NVP_NELEM(nvp);
2202         nvlist_t **nvlp = EMBEDDED_NVL_ARRAY(nvp);
2203         int i;

2205         switch (nvs->nvs_op) {
2206         case NVS_OP_ENCODE:
2207                 for (i = 0; i < nelem; i++)
2208                         if (nvs_embedded(nvs, nvlp[i]) != 0)
2209                                 return (EFAULT);
2210                 break;

2212         case NVS_OP_DECODE: {
2213                 size_t len = nelem * sizeof (uint64_t);
2214                 nvlist_t *embedded = (nvlist_t *)((uintptr_t)nvlp + len);

2216                 bzero(nvlp, len);       /* don't trust packed data */
2217                 for (i = 0; i < nelem; i++) {
2218                         if (nvs_embedded(nvs, embedded) != 0) {
2219                                 nvpair_free(nvp);
2220                                 return (EFAULT);
2221                         }

2223                         nvlp[i] = embedded++;
2224                 }
2225                 break;
2226         }
2227         case NVS_OP_GETSIZE: {
2228                 uint64_t nvsize = 0;

2230                 for (i = 0; i < nelem; i++) {
2231                         size_t nvp_sz = 0;

2233                         if (nvs_operation(nvs, nvlp[i], &nvp_sz) != 0)
2234                                 return (EINVAL);

2236                         if ((nvsize += nvp_sz) > INT32_MAX)
2237                                 return (EINVAL);
2238                 }
```

```
2240                 *size = nvsize;
2241                 break;
2242         }
2243         default:
2244                 return (EINVAL);
2245         }

2247         return (0);
2248 }

2250 static int nvs_native(nvstream_t *, nvlist_t *, char *, size_t *);
2251 static int nvs_xdr(nvstream_t *, nvlist_t *, char *, size_t *);

2253 /*
2254  * Common routine for nvlist operations:
2255  * encode, decode, getsize (encoded size).
2256  */
2257 static int
2258 nvlist_common(nvlist_t *nvl, char *buf, size_t *buflen, int encoding,
2259     int nvs_op)
2260 {
2261         int err = 0;
2262         nvstream_t nvs;
2263         int nvl_endian;
2264 #ifdef  _LITTLE_ENDIAN
2265         int host_endian = 1;
2266 #else
2267         int host_endian = 0;
2268 #endif  /* _LITTLE_ENDIAN */
2269         nvs_header_t *nvh = (void *)buf;

2271         if (buflen == NULL || nvl == NULL ||
2272             (nvs.nvs_priv = (nvpriv_t *)(uintptr_t)nvl->nvl_priv) == NULL)
2273                 return (EINVAL);

2275         nvs.nvs_op = nvs_op;

2277         /*
2278          * For NVS_OP_ENCODE and NVS_OP_DECODE make sure an nvlist and
2279          * a buffer is allocated.  The first 4 bytes in the buffer are
2280          * used for encoding method and host endian.
2281          */
2282         switch (nvs_op) {
2283         case NVS_OP_ENCODE:
2284                 if (buf == NULL || *buflen < sizeof (nvs_header_t))
2285                         return (EINVAL);

2287                 nvh->nvh_encoding = encoding;
2288                 nvh->nvh_endian = nvl_endian = host_endian;
2289                 nvh->nvh_reserved1 = 0;
2290                 nvh->nvh_reserved2 = 0;
2291                 break;

2293         case NVS_OP_DECODE:
2294                 if (buf == NULL || *buflen < sizeof (nvs_header_t))
2295                         return (EINVAL);

2297                 /* get method of encoding from first byte */
2298                 encoding = nvh->nvh_encoding;
2299                 nvl_endian = nvh->nvh_endian;
2300                 break;

2302         case NVS_OP_GETSIZE:
2303                 nvl_endian = host_endian;

2305                 /*
```



```
2306                  * add the size for encoding
2307                  */
2308                 *buflen = sizeof (nvs_header_t);
2309                 break;

2311         default:
2312                 return (ENOTSUP);
2313         }

2315         /*
2316          * Create an nvstream with proper encoding method
2317          */
2318         switch (encoding) {
2319         case NV_ENCODE_NATIVE:
2320                 /*
2321                  * check endianness, in case we are unpacking
2322                  * from a file
2323                  */
2324                 if (nvl_endian != host_endian)
2325                         return (ENOTSUP);
2326                 err = nvs_native(&nvs, nvl, buf, buflen);
2327                 break;
2328         case NV_ENCODE_XDR:
2329                 err = nvs_xdr(&nvs, nvl, buf, buflen);
2330                 break;
2331         default:
2332                 err = ENOTSUP;
2333                 break;
2334         }

2336         return (err);
2337 }

2339 int
2340 nvlist_size(nvlist_t *nvl, size_t *size, int encoding)
2341 {
2342         return (nvlist_common(nvl, NULL, size, encoding, NVS_OP_GETSIZE));
2343 }

2345 /*
2346  * Pack nvlist into contiguous memory
2347  */
2348 /*ARGSUSED1*/
2349 int
2350 nvlist_pack(nvlist_t *nvl, char **bufp, size_t *buflen, int encoding,
2351     int kmflag)
2352 {
2353 #if defined(_KERNEL) && !defined(_BOOT)
2354         return (nvlist_xpack(nvl, bufp, buflen, encoding,
2355             (kmflag == KM_SLEEP ? nv_alloc_sleep : nv_alloc_nosleep)));
2356 #else
2357         return (nvlist_xpack(nvl, bufp, buflen, encoding, nv_alloc_nosleep));
2358 #endif
2359 }

2361 int
2362 nvlist_xpack(nvlist_t *nvl, char **bufp, size_t *buflen, int encoding,
2363     nv_alloc_t *nva)
2364 {
2365         nvpriv_t nvpriv;
2366         size_t alloc_size;
2367         char *buf;
2368         int err;

2370         if (nva == NULL || nvl == NULL || bufp == NULL || buflen == NULL)
2371                 return (EINVAL);
```

```
2373         if (*bufp != NULL)
2374                 return (nvlist_common(nvl, *bufp, buflen, encoding,
2375                     NVS_OP_ENCODE));
2376
2377         /*
2378          * Here is a difficult situation:
2379          * 1. The nvlist has fixed allocator properties.
2380          *    All other nvlist routines (like nvlist_add_*, ...) use
2381          *    these properties.
2382          * 2. When using nvlist_pack() the user can specify his own
2383          *    allocator properties (e.g. by using KM_NOSLEEP).
2384          *
2385          * We use the user specified properties (2). A clearer solution
2386          * will be to remove the kmflag from nvlist_pack(), but we will
2387          * not change the interface.
2388          */
2389         nv_priv_init(&nvpriv, nva, 0);
2390
2391         if (err = nvlist_size(nvl, &alloc_size, encoding))
2392                 return (err);
2393
2394         if ((buf = nv_mem_zalloc(&nvpriv, alloc_size)) == NULL)
2395                 return (ENOMEM);
2396
2397         if ((err = nvlist_common(nvl, buf, &alloc_size, encoding,
2398             NVS_OP_ENCODE)) != 0) {
2399                 nv_mem_free(&nvpriv, buf, alloc_size);
2400         } else {
2401                 *buflen = alloc_size;
2402                 *bufp = buf;
2403         }
2404
2405         return (err);
2406 }
2407
2408 /*
2409  * Unpack buf into an nvlist_t
2410  */
2411 /*ARGSUSED1*/
2412 int
2413 nvlist_unpack(char *buf, size_t buflen, nvlist_t **nvlp, int kmflag)
2414 {
2415 #if defined(_KERNEL) && !defined(_BOOT)
2416         return (nvlist_xunpack(buf, buflen, nvlp,
2417             (kmflag == KM_SLEEP ? nv_alloc_sleep : nv_alloc_nosleep)));
2418 #else
2419         return (nvlist_xunpack(buf, buflen, nvlp, nv_alloc_nosleep));
2420 #endif
2421 }
2422
2423 int
2424 nvlist_xunpack(char *buf, size_t buflen, nvlist_t **nvlp, nv_alloc_t *nva)
2425 {
2426         nvlist_t *nvl;
2427         int err;
2428
2429         if (nvlp == NULL)
2430                 return (EINVAL);
2431
2432         if ((err = nvlist_xalloc(&nvl, 0, nva)) != 0)
2433                 return (err);
2434
2435         if ((err = nvlist_common(nvl, buf, &buflen, 0, NVS_OP_DECODE)) != 0)
2436                 nvlist_free(nvl);
2437         else
```



```
2438                 *nvlp = nvl;
2439
2440         return (err);
2441 }
2442
2443 /*
2444  * Native encoding functions
2445  */
2446 typedef struct {
2447         /*
2448          * This structure is used when decoding a packed nvpair in
2449          * the native format.  n_base points to a buffer containing the
2450          * packed nvpair.  n_end is a pointer to the end of the buffer.
2451          * (n_end actually points to the first byte past the end of the
2452          * buffer.)  n_curr is a pointer that lies between n_base and n_end.
2453          * It points to the current data that we are decoding.
2454          * The amount of data left in the buffer is equal to n_end - n_curr.
2455          * n_flag is used to recognize a packed embedded list.
2456          */
2457         caddr_t n_base;
2458         caddr_t n_end;
2459         caddr_t n_curr;
2460         uint_t  n_flag;
2461 } nvs_native_t;
2462
2463 static int
2464 nvs_native_create(nvstream_t *nvs, nvs_native_t *native, char *buf,
2465     size_t buflen)
2466 {
2467         switch (nvs->nvs_op) {
2468         case NVS_OP_ENCODE:
2469         case NVS_OP_DECODE:
2470                 nvs->nvs_private = native;
2471                 native->n_curr = native->n_base = buf;
2472                 native->n_end = buf + buflen;
2473                 native->n_flag = 0;
2474                 return (0);
2475
2476         case NVS_OP_GETSIZE:
2477                 nvs->nvs_private = native;
2478                 native->n_curr = native->n_base = native->n_end = NULL;
2479                 native->n_flag = 0;
2480                 return (0);
2481         default:
2482                 return (EINVAL);
2483         }
2484 }
2485
2486 /*ARGSUSED*/
2487 static void
2488 nvs_native_destroy(nvstream_t *nvs)
2489 {
2490 }
2491
2492 static int
2493 native_cp(nvstream_t *nvs, void *buf, size_t size)
2494 {
2495         nvs_native_t *native = (nvs_native_t *)nvs->nvs_private;
2496
2497         if (native->n_curr + size > native->n_end)
2498                 return (EFAULT);
2499
2500         /*
2501          * The bcopy() below eliminates alignment requirement
2502          * on the buffer (stream) and is preferred over direct access.
2503          */
```

```
2504         switch (nvs->nvs_op) {
2505         case NVS_OP_ENCODE:
2506                 bcopy(buf, native->n_curr, size);
2507                 break;
2508         case NVS_OP_DECODE:
2509                 bcopy(native->n_curr, buf, size);
2510                 break;
2511         default:
2512                 return (EINVAL);
2513         }

2515         native->n_curr += size;
2516         return (0);
2517 }

2519 /*
2520  * operate on nvlist_t header
2521  */
2522 static int
2523 nvs_native_nvlist(nvstream_t *nvs, nvlist_t *nvl, size_t *size)
2524 {
2525         nvs_native_t *native = nvs->nvs_private;

2527         switch (nvs->nvs_op) {
2528         case NVS_OP_ENCODE:
2529         case NVS_OP_DECODE:
2530                 if (native->n_flag)
2531                         return (0);     /* packed embedded list */

2533                 native->n_flag = 1;

2535                 /* copy version and nvflag of the nvlist_t */
2536                 if (native_cp(nvs, &nvl->nvl_version, sizeof (int32_t)) != 0 ||
2537                     native_cp(nvs, &nvl->nvl_nvflag, sizeof (int32_t)) != 0)
2538                         return (EFAULT);

2540                 return (0);

2542         case NVS_OP_GETSIZE:
2543                 /*
2544                  * if calculate for packed embedded list
2545                  *      4 for end of the embedded list
2546                  * else
2547                  *      2 * sizeof (int32_t) for nvl_version and nvl_nvflag
2548                  *      and 4 for end of the entire list
2549                  */
2550                 if (native->n_flag) {
2551                         *size += 4;
2552                 } else {
2553                         native->n_flag = 1;
2554                         *size += 2 * sizeof (int32_t) + 4;
2555                 }

2557                 return (0);

2559         default:
2560                 return (EINVAL);
2561         }
2562 }

2564 static int
2565 nvs_native_nvl_fini(nvstream_t *nvs)
2566 {
2567         if (nvs->nvs_op == NVS_OP_ENCODE) {
2568                 nvs_native_t *native = (nvs_native_t *)nvs->nvs_private;
2569                 /*
```



```
2570                  * Add 4 zero bytes at end of nvlist. They are used
2571                  * for end detection by the decode routine.
2572                  */
2573                 if (native->n_curr + sizeof (int) > native->n_end)
2574                         return (EFAULT);

2576                 bzero(native->n_curr, sizeof (int));
2577                 native->n_curr += sizeof (int);
2578         }

2580         return (0);
2581 }

2583 static int
2584 nvpair_native_embedded(nvstream_t *nvs, nvpair_t *nvp)
2585 {
2586         if (nvs->nvs_op == NVS_OP_ENCODE) {
2587                 nvs_native_t *native = (nvs_native_t *)nvs->nvs_private;
2588                 nvlist_t *packed = (void *)
2589                     (native->n_curr - nvp->nvp_size + NVP_VALOFF(nvp));
2590                 /*
2591                  * Null out the pointer that is meaningless in the packed
2592                  * structure. The address may not be aligned, so we have
2593                  * to use bzero.
2594                  */
2595                 bzero(&packed->nvl_priv, sizeof (packed->nvl_priv));
2596         }

2598         return (nvs_embedded(nvs, EMBEDDED_NVL(nvp)));
2599 }

2601 static int
2602 nvpair_native_embedded_array(nvstream_t *nvs, nvpair_t *nvp)
2603 {
2604         if (nvs->nvs_op == NVS_OP_ENCODE) {
2605                 nvs_native_t *native = (nvs_native_t *)nvs->nvs_private;
2606                 char *value = native->n_curr - nvp->nvp_size + NVP_VALOFF(nvp);
2607                 size_t len = NVP_NELEM(nvp) * sizeof (uint64_t);
2608                 nvlist_t *packed = (nvlist_t *)((uintptr_t)value + len);
2609                 int i;
2610                 /*
2611                  * Null out pointers that are meaningless in the packed
2612                  * structure. The addresses may not be aligned, so we have
2613                  * to use bzero.
2614                  */
2615                 bzero(value, len);

2617                 for (i = 0; i < NVP_NELEM(nvp); i++, packed++)
2618                         /*
2619                          * Null out the pointer that is meaningless in the
2620                          * packed structure. The address may not be aligned,
2621                          * so we have to use bzero.
2622                          */
2623                         bzero(&packed->nvl_priv, sizeof (packed->nvl_priv));
2624         }

2626         return (nvs_embedded_nvl_array(nvs, nvp, NULL));
2627 }

2629 static void
2630 nvpair_native_string_array(nvstream_t *nvs, nvpair_t *nvp)
2631 {
2632         switch (nvs->nvs_op) {
2633         case NVS_OP_ENCODE: {
2634                 nvs_native_t *native = (nvs_native_t *)nvs->nvs_private;
2635                 uint64_t *strp = (void *)
```

```
2636                    (native->n_curr - nvp->nvp_size + NVP_VALOFF(nvp));
2637                /*
2638                 * Null out pointers that are meaningless in the packed
2639                 * structure. The addresses may not be aligned, so we have
2640                 * to use bzero.
2641                 */
2642                bzero(strp, NVP_NELEM(nvp) * sizeof (uint64_t));
2643                break;
2644        }
2645        case NVS_OP_DECODE: {
2646                char **strp = (void *)NVP_VALUE(nvp);
2647                char *buf = ((char *)strp + NVP_NELEM(nvp) * sizeof (uint64_t));
2648                int i;

2650                for (i = 0; i < NVP_NELEM(nvp); i++) {
2651                        strp[i] = buf;
2652                        buf += strlen(buf) + 1;
2653                }
2654                break;
2655        }
2656        }
2657 }

2659 static int
2660 nvs_native_nvp_op(nvstream_t *nvs, nvpair_t *nvp)
2661 {
2662        data_type_t type;
2663        int value_sz;
2664        int ret = 0;

2666        /*
2667         * We do the initial bcopy of the data before we look at
2668         * the nvpair type, because when we're decoding, we won't
2669         * have the correct values for the pair until we do the bcopy.
2670         */
2671        switch (nvs->nvs_op) {
2672        case NVS_OP_ENCODE:
2673        case NVS_OP_DECODE:
2674                if (native_cp(nvs, nvp, nvp->nvp_size) != 0)
2675                        return (EFAULT);
2676                break;
2677        default:
2678                return (EINVAL);
2679        }

2681        /* verify nvp_name_sz, check the name string length */
2682        if (i_validate_nvpair_name(nvp) != 0)
2683                return (EFAULT);

2685        type = NVP_TYPE(nvp);

2687        /*
2688         * Verify type and nelem and get the value size.
2689         * In case of data types DATA_TYPE_STRING and DATA_TYPE_STRING_ARRAY
2690         * is the size of the string(s) excluded.
2691         */
2692        if ((value_sz = i_get_value_size(type, NULL, NVP_NELEM(nvp))) < 0)
2693                return (EFAULT);

2695        if (NVP_SIZE_CALC(nvp->nvp_name_sz, value_sz) > nvp->nvp_size)
2696                return (EFAULT);

2698        switch (type) {
2699        case DATA_TYPE_NVLIST:
2700                ret = nvpair_native_embedded(nvs, nvp);
2701                break;
```



```
2702        case DATA_TYPE_NVLIST_ARRAY:
2703                ret = nvpair_native_embedded_array(nvs, nvp);
2704                break;
2705        case DATA_TYPE_STRING_ARRAY:
2706                nvpair_native_string_array(nvs, nvp);
2707                break;
2708        default:
2709                break;
2710        }

2712        return (ret);
2713 }

2715 static int
2716 nvs_native_nvp_size(nvstream_t *nvs, nvpair_t *nvp, size_t *size)
2717 {
2718        uint64_t nvp_sz = nvp->nvp_size;

2720        switch (NVP_TYPE(nvp)) {
2721        case DATA_TYPE_NVLIST: {
2722                size_t nvsize = 0;

2724                if (nvs_operation(nvs, EMBEDDED_NVL(nvp), &nvsize) != 0)
2725                        return (EINVAL);

2727                nvp_sz += nvsize;
2728                break;
2729        }
2730        case DATA_TYPE_NVLIST_ARRAY: {
2731                size_t nvsize;

2733                if (nvs_embedded_nvl_array(nvs, nvp, &nvsize) != 0)
2734                        return (EINVAL);

2736                nvp_sz += nvsize;
2737                break;
2738        }
2739        default:
2740                break;
2741        }

2743        if (nvp_sz > INT32_MAX)
2744                return (EINVAL);

2746        *size = nvp_sz;

2748        return (0);
2749 }

2751 static int
2752 nvs_native_nvpair(nvstream_t *nvs, nvpair_t *nvp, size_t *size)
2753 {
2754        switch (nvs->nvs_op) {
2755        case NVS_OP_ENCODE:
2756                return (nvs_native_nvp_op(nvs, nvp));

2758        case NVS_OP_DECODE: {
2759                nvs_native_t *native = (nvs_native_t *)nvs->nvs_private;
2760                int32_t decode_len;

2762                /* try to read the size value from the stream */
2763                if (native->n_curr + sizeof (int32_t) > native->n_end)
2764                        return (EFAULT);
2765                bcopy(native->n_curr, &decode_len, sizeof (int32_t));

2767                /* sanity check the size value */
```

```
2768                 if (decode_len < 0 ||
2769                     decode_len > native->n_end - native->n_curr)
2770                         return (EFAULT);

2772                 *size = decode_len;

2774                 /*
2775                  * If at the end of the stream then move the cursor
2776                  * forward, otherwise nvpair_native_op() will read
2777                  * the entire nvpair at the same cursor position.
2778                  */
2779                 if (*size == 0)
2780                         native->n_curr += sizeof (int32_t);
2781                 break;
2782         }

2784         default:
2785                 return (EINVAL);
2786         }

2788         return (0);
2789 }

2791 static const nvs_ops_t nvs_native_ops = {
2792         nvs_native_nvlist,
2793         nvs_native_nvpair,
2794         nvs_native_nvp_op,
2795         nvs_native_nvp_size,
2796         nvs_native_nvl_fini
2797 };

2799 static int
2800 nvs_native(nvstream_t *nvs, nvlist_t *nvl, char *buf, size_t *buflen)
2801 {
2802         nvs_native_t native;
2803         int err;

2805         nvs->nvs_ops = &nvs_native_ops;

2807         if ((err = nvs_native_create(nvs, &native, buf + sizeof (nvs_header_t),
2808             *buflen - sizeof (nvs_header_t))) != 0)
2809                 return (err);

2811         err = nvs_operation(nvs, nvl, buflen);

2813         nvs_native_destroy(nvs);

2815         return (err);
2816 }

2818 /*
2819  * XDR encoding functions
2820  *
2821  * An xdr packed nvlist is encoded as:
2822  *
2823  *   - encoding methode and host endian (4 bytes)
2824  *   - nvl_version (4 bytes)
2825  *   - nvl_nvflag (4 bytes)
2826  *
2827  *   - encoded nvpairs, the format of one xdr encoded nvpair is:
2828  *       - encoded size of the nvpair (4 bytes)
2829  *       - decoded size of the nvpair (4 bytes)
2830  *       - name string, (4 + sizeof(NV_ALIGN4(string))
2831  *         a string is coded as size (4 bytes) and data
2832  *       - data type (4 bytes)
2833  *       - number of elements in the nvpair (4 bytes)
```



```
2834  *       - data
2835  *
2836  *   - 2 zero's for end of the entire list (8 bytes)
2837  */
2838 static int
2839 nvs_xdr_create(nvstream_t *nvs, XDR *xdr, char *buf, size_t buflen)
2840 {
2841         /* xdr data must be 4 byte aligned */
2842         if ((ulong_t)buf % 4 != 0)
2843                 return (EFAULT);

2845         switch (nvs->nvs_op) {
2846         case NVS_OP_ENCODE:
2847                 xdrmem_create(xdr, buf, (uint_t)buflen, XDR_ENCODE);
2848                 nvs->nvs_private = xdr;
2849                 return (0);
2850         case NVS_OP_DECODE:
2851                 xdrmem_create(xdr, buf, (uint_t)buflen, XDR_DECODE);
2852                 nvs->nvs_private = xdr;
2853                 return (0);
2854         case NVS_OP_GETSIZE:
2855                 nvs->nvs_private = NULL;
2856                 return (0);
2857         default:
2858                 return (EINVAL);
2859         }
2860 }

2862 static void
2863 nvs_xdr_destroy(nvstream_t *nvs)
2864 {
2865         switch (nvs->nvs_op) {
2866         case NVS_OP_ENCODE:
2867         case NVS_OP_DECODE:
2868                 xdr_destroy((XDR *)nvs->nvs_private);
2869                 break;
2870         default:
2871                 break;
2872         }
2873 }

2875 static int
2876 nvs_xdr_nvlist(nvstream_t *nvs, nvlist_t *nvl, size_t *size)
2877 {
2878         switch (nvs->nvs_op) {
2879         case NVS_OP_ENCODE:
2880         case NVS_OP_DECODE: {
2881                 XDR     *xdr = nvs->nvs_private;

2883                 if (!xdr_int(xdr, &nvl->nvl_version) ||
2884                     !xdr_u_int(xdr, &nvl->nvl_nvflag))
2885                         return (EFAULT);
2886                 break;
2887         }
2888         case NVS_OP_GETSIZE: {
2889                 /*
2890                  * 2 * 4 for nvl_version + nvl_nvflag
2891                  * and 8 for end of the entire list
2892                  */
2893                 *size += 2 * 4 + 8;
2894                 break;
2895         }
2896         default:
2897                 return (EINVAL);
2898         }
2899         return (0);
```

```
2900 }

2902 static int
2903 nvs_xdr_nvl_fini(nvstream_t *nvs)
2904 {
2905         if (nvs->nvs_op == NVS_OP_ENCODE) {
2906                 XDR *xdr = nvs->nvs_private;
2907                 int zero = 0;

2909                 if (!xdr_int(xdr, &zero) || !xdr_int(xdr, &zero))
2910                         return (EFAULT);
2911         }

2913         return (0);
2914 }

2916 /*
2917  * The format of xdr encoded nvpair is:
2918  * encode_size, decode_size, name string, data type, nelem, data
2919  */
2920 static int
2921 nvs_xdr_nvp_op(nvstream_t *nvs, nvpair_t *nvp)
2922 {
2923         data_type_t type;
2924         char    *buf;
2925         char    *buf_end = (char *)nvp + nvp->nvp_size;
2926         int     value_sz;
2927         uint_t  nelem, buflen;
2928         bool_t  ret = FALSE;
2929         XDR     *xdr = nvs->nvs_private;

2931         ASSERT(xdr != NULL && nvp != NULL);

2933         /* name string */
2934         if ((buf = NVP_NAME(nvp)) >= buf_end)
2935                 return (EFAULT);
2936         buflen = buf_end - buf;

2938         if (!xdr_string(xdr, &buf, buflen - 1))
2939                 return (EFAULT);
2940         nvp->nvp_name_sz = strlen(buf) + 1;

2942         /* type and nelem */
2943         if (!xdr_int(xdr, (int *)&nvp->nvp_type) ||
2944             !xdr_int(xdr, &nvp->nvp_value_elem))
2945                 return (EFAULT);

2947         type = NVP_TYPE(nvp);
2948         nelem = nvp->nvp_value_elem;

2950         /*
2951          * Verify type and nelem and get the value size.
2952          * In case of data types DATA_TYPE_STRING and DATA_TYPE_STRING_ARRAY
2953          * is the size of the string(s) excluded.
2954          */
2955         if ((value_sz = i_get_value_size(type, NULL, nelem)) < 0)
2956                 return (EFAULT);

2958         /* if there is no data to extract then return */
2959         if (nelem == 0)
2960                 return (0);

2962         /* value */
2963         if ((buf = NVP_VALUE(nvp)) >= buf_end)
2964                 return (EFAULT);
2965         buflen = buf_end - buf;
```



```
2967         if (buflen < value_sz)
2968                 return (EFAULT);

2970         switch (type) {
2971         case DATA_TYPE_NVLIST:
2972                 if (nvs_embedded(nvs, (void *)buf) == 0)
2973                         return (0);
2974                 break;

2976         case DATA_TYPE_NVLIST_ARRAY:
2977                 if (nvs_embedded_nvl_array(nvs, nvp, NULL) == 0)
2978                         return (0);
2979                 break;

2981         case DATA_TYPE_BOOLEAN:
2982                 ret = TRUE;
2983                 break;

2985         case DATA_TYPE_BYTE:
2986         case DATA_TYPE_INT8:
2987         case DATA_TYPE_UINT8:
2988                 ret = xdr_char(xdr, buf);
2989                 break;

2991         case DATA_TYPE_INT16:
2992                 ret = xdr_short(xdr, (void *)buf);
2993                 break;

2995         case DATA_TYPE_UINT16:
2996                 ret = xdr_u_short(xdr, (void *)buf);
2997                 break;

2999         case DATA_TYPE_BOOLEAN_VALUE:
3000         case DATA_TYPE_INT32:
3001                 ret = xdr_int(xdr, (void *)buf);
3002                 break;

3004         case DATA_TYPE_UINT32:
3005                 ret = xdr_u_int(xdr, (void *)buf);
3006                 break;

3008         case DATA_TYPE_INT64:
3009                 ret = xdr_longlong_t(xdr, (void *)buf);
3010                 break;

3012         case DATA_TYPE_UINT64:
3013                 ret = xdr_u_longlong_t(xdr, (void *)buf);
3014                 break;

3016         case DATA_TYPE_HRTIME:
3017                 /*
3018                  * NOTE: must expose the definition of hrtime_t here
3019                  */
3020                 ret = xdr_longlong_t(xdr, (void *)buf);
3021                 break;
3022 #if !defined(_KERNEL)
3023         case DATA_TYPE_DOUBLE:
3024                 ret = xdr_double(xdr, (void *)buf);
3025                 break;
3026 #endif
3027         case DATA_TYPE_STRING:
3028                 ret = xdr_string(xdr, &buf, buflen - 1);
3029                 break;

3031         case DATA_TYPE_BYTE_ARRAY:
```

```
3032                ret = xdr_opaque(xdr, buf, nelem);
3033                break;

3035        case DATA_TYPE_INT8_ARRAY:
3036        case DATA_TYPE_UINT8_ARRAY:
3037                ret = xdr_array(xdr, &buf, &nelem, buflen, sizeof (int8_t),
3038                    (xdrproc_t)xdr_char);
3039                break;

3041        case DATA_TYPE_INT16_ARRAY:
3042                ret = xdr_array(xdr, &buf, &nelem, buflen / sizeof (int16_t),
3043                    sizeof (int16_t), (xdrproc_t)xdr_short);
3044                break;

3046        case DATA_TYPE_UINT16_ARRAY:
3047                ret = xdr_array(xdr, &buf, &nelem, buflen / sizeof (uint16_t),
3048                    sizeof (uint16_t), (xdrproc_t)xdr_u_short);
3049                break;

3051        case DATA_TYPE_BOOLEAN_ARRAY:
3052        case DATA_TYPE_INT32_ARRAY:
3053                ret = xdr_array(xdr, &buf, &nelem, buflen / sizeof (int32_t),
3054                    sizeof (int32_t), (xdrproc_t)xdr_int);
3055                break;

3057        case DATA_TYPE_UINT32_ARRAY:
3058                ret = xdr_array(xdr, &buf, &nelem, buflen / sizeof (uint32_t),
3059                    sizeof (uint32_t), (xdrproc_t)xdr_u_int);
3060                break;

3062        case DATA_TYPE_INT64_ARRAY:
3063                ret = xdr_array(xdr, &buf, &nelem, buflen / sizeof (int64_t),
3064                    sizeof (int64_t), (xdrproc_t)xdr_longlong_t);
3065                break;

3067        case DATA_TYPE_UINT64_ARRAY:
3068                ret = xdr_array(xdr, &buf, &nelem, buflen / sizeof (uint64_t),
3069                    sizeof (uint64_t), (xdrproc_t)xdr_u_longlong_t);
3070                break;

3072        case DATA_TYPE_STRING_ARRAY: {
3073                size_t len = nelem * sizeof (uint64_t);
3074                char **strp = (void *)buf;
3075                int i;

3077                if (nvs->nvs_op == NVS_OP_DECODE)
3078                        bzero(buf, len);        /* don't trust packed data */

3080                for (i = 0; i < nelem; i++) {
3081                        if (buflen <= len)
3082                                return (EFAULT);

3084                        buf += len;
3085                        buflen -= len;

3087                        if (xdr_string(xdr, &buf, buflen - 1) != TRUE)
3088                                return (EFAULT);

3090                        if (nvs->nvs_op == NVS_OP_DECODE)
3091                                strp[i] = buf;
3092                        len = strlen(buf) + 1;
3093                }
3094                ret = TRUE;
3095                break;
3096        }
3097        default:
```



```
3098                break;
3099        }

3101        return (ret == TRUE ? 0 : EFAULT);
3102 }

3104 static int
3105 nvs_xdr_nvp_size(nvstream_t *nvs, nvpair_t *nvp, size_t *size)
3106 {
3107        data_type_t type = NVP_TYPE(nvp);
3108        /*
3109         * encode_size + decode_size + name string size + data type + nelem
3110         * where name string size = 4 + NV_ALIGN4(strlen(NVP_NAME(nvp)))
3111         */
3112        uint64_t nvp_sz = 4 + 4 + 4 + NV_ALIGN4(strlen(NVP_NAME(nvp))) + 4 + 4;

3114        switch (type) {
3115        case DATA_TYPE_BOOLEAN:
3116                break;

3118        case DATA_TYPE_BOOLEAN_VALUE:
3119        case DATA_TYPE_BYTE:
3120        case DATA_TYPE_INT8:
3121        case DATA_TYPE_UINT8:
3122        case DATA_TYPE_INT16:
3123        case DATA_TYPE_UINT16:
3124        case DATA_TYPE_INT32:
3125        case DATA_TYPE_UINT32:
3126                nvp_sz += 4;    /* 4 is the minimum xdr unit */
3127                break;

3129        case DATA_TYPE_INT64:
3130        case DATA_TYPE_UINT64:
3131        case DATA_TYPE_HRTIME:
3132 #if !defined(_KERNEL)
3133        case DATA_TYPE_DOUBLE:
3134 #endif
3135                nvp_sz += 8;
3136                break;

3138        case DATA_TYPE_STRING:
3139                nvp_sz += 4 + NV_ALIGN4(strlen((char *)NVP_VALUE(nvp)));
3140                break;

3142        case DATA_TYPE_BYTE_ARRAY:
3143                nvp_sz += NV_ALIGN4(NVP_NELEM(nvp));
3144                break;

3146        case DATA_TYPE_BOOLEAN_ARRAY:
3147        case DATA_TYPE_INT8_ARRAY:
3148        case DATA_TYPE_UINT8_ARRAY:
3149        case DATA_TYPE_INT16_ARRAY:
3150        case DATA_TYPE_UINT16_ARRAY:
3151        case DATA_TYPE_INT32_ARRAY:
3152        case DATA_TYPE_UINT32_ARRAY:
3153                nvp_sz += 4 + 4 * (uint64_t)NVP_NELEM(nvp);
3154                break;

3156        case DATA_TYPE_INT64_ARRAY:
3157        case DATA_TYPE_UINT64_ARRAY:
3158                nvp_sz += 4 + 8 * (uint64_t)NVP_NELEM(nvp);
3159                break;

3161        case DATA_TYPE_STRING_ARRAY: {
3162                int i;
3163                char **strs = (void *)NVP_VALUE(nvp);
```

```
3165                 for (i = 0; i < NVP_NELEM(nvp); i++)
3166                         nvp_sz += 4 + NV_ALIGN4(strlen(strs[i]));
3167
3168                 break;
3169         }
3170
3171         case DATA_TYPE_NVLIST:
3172         case DATA_TYPE_NVLIST_ARRAY: {
3173                 size_t nvsize = 0;
3174                 int old_nvs_op = nvs->nvs_op;
3175                 int err;
3176
3177                 nvs->nvs_op = NVS_OP_GETSIZE;
3178                 if (type == DATA_TYPE_NVLIST)
3179                         err = nvs_operation(nvs, EMBEDDED_NVL(nvp), &nvsize);
3180                 else
3181                         err = nvs_embedded_nvl_array(nvs, nvp, &nvsize);
3182                 nvs->nvs_op = old_nvs_op;
3183
3184                 if (err != 0)
3185                         return (EINVAL);
3186
3187                 nvp_sz += nvsize;
3188                 break;
3189         }
3190
3191         default:
3192                 return (EINVAL);
3193         }
3194
3195         if (nvp_sz > INT32_MAX)
3196                 return (EINVAL);
3197
3198         *size = nvp_sz;
3199
3200         return (0);
3201 }
3202
3203
3204 /*
3205  * The NVS_XDR_MAX_LEN macro takes a packed xdr buffer of size x and estimates
3206  * the largest nvpair that could be encoded in the buffer.
3207  *
3208  * See comments above nvpair_xdr_op() for the format of xdr encoding.
3209  * The size of a xdr packed nvpair without any data is 5 words.
3210  *
3211  * Using the size of the data directly as an estimate would be ok
3212  * in all cases except one.  If the data type is of DATA_TYPE_STRING_ARRAY
3213  * then the actual nvpair has space for an array of pointers to index
3214  * the strings.  These pointers are not encoded into the packed xdr buffer.
3215  *
3216  * If the data is of type DATA_TYPE_STRING_ARRAY and all the strings are
3217  * of length 0, then each string is endcoded in xdr format as a single word.
3218  * Therefore when expanded to an nvpair there will be 2.25 word used for
3219  * each string.  (a int64_t allocated for pointer usage, and a single char
3220  * for the null termination.)
3221  *
3222  * This is the calculation performed by the NVS_XDR_MAX_LEN macro.
3223  */
3224 #define NVS_XDR_HDR_LEN         ((size_t)(5 * 4))
3225 #define NVS_XDR_DATA_LEN(y)     (((size_t)(y) <= NVS_XDR_HDR_LEN) ? \
3226                                     0 : ((size_t)(y) - NVS_XDR_HDR_LEN))
3227 #define NVS_XDR_MAX_LEN(x)      (NVP_SIZE_CALC(1, 0) + \
3228                                         (NVS_XDR_DATA_LEN(x) * 2) + \
3229                                         NV_ALIGN4((NVS_XDR_DATA_LEN(x) / 4)))
```



```
3231 static int
3232 nvs_xdr_nvpair(nvstream_t *nvs, nvpair_t *nvp, size_t *size)
3233 {
3234         XDR     *xdr = nvs->nvs_private;
3235         int32_t encode_len, decode_len;
3236
3237         switch (nvs->nvs_op) {
3238         case NVS_OP_ENCODE: {
3239                 size_t nvsize;
3240
3241                 if (nvs_xdr_nvp_size(nvs, nvp, &nvsize) != 0)
3242                         return (EFAULT);
3243
3244                 decode_len = nvp->nvp_size;
3245                 encode_len = nvsize;
3246                 if (!xdr_int(xdr, &encode_len) || !xdr_int(xdr, &decode_len))
3247                         return (EFAULT);
3248
3249                 return (nvs_xdr_nvp_op(nvs, nvp));
3250         }
3251         case NVS_OP_DECODE: {
3252                 struct xdr_bytesrec bytesrec;
3253
3254                 /* get the encode and decode size */
3255                 if (!xdr_int(xdr, &encode_len) || !xdr_int(xdr, &decode_len))
3256                         return (EFAULT);
3257                 *size = decode_len;
3258
3259                 /* are we at the end of the stream? */
3260                 if (*size == 0)
3261                         return (0);
3262
3263                 /* sanity check the size parameter */
3264                 if (!xdr_control(xdr, XDR_GET_BYTES_AVAIL, &bytesrec))
3265                         return (EFAULT);
3266
3267                 if (*size > NVS_XDR_MAX_LEN(bytesrec.xc_num_avail))
3268                         return (EFAULT);
3269                 break;
3270         }
3271
3272         default:
3273                 return (EINVAL);
3274         }
3275         return (0);
3276 }
3277
3278 static const struct nvs_ops nvs_xdr_ops = {
3279         nvs_xdr_nvlist,
3280         nvs_xdr_nvpair,
3281         nvs_xdr_nvp_op,
3282         nvs_xdr_nvp_size,
3283         nvs_xdr_nvl_fini
3284 };
3285
3286 static int
3287 nvs_xdr(nvstream_t *nvs, nvlist_t *nvl, char *buf, size_t *buflen)
3288 {
3289         XDR xdr;
3290         int err;
3291
3292         nvs->nvs_ops = &nvs_xdr_ops;
3293
3294         if ((err = nvs_xdr_create(nvs, &xdr, buf + sizeof (nvs_header_t),
3295             *buflen - sizeof (nvs_header_t))) != 0)
```

```
3296                 return (err);

3298         err = nvs_operation(nvs, nvl, buflen);

3300         nvs_xdr_destroy(nvs);

3302         return (err);
3303 }
```

```
**********************************************************
    2697 Thu Sep  5 23:02:08 2013
new/usr/src/head/iso/stddef_iso.h
3373 gcc >= 4.5 concerns about offsetof()
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*      Copyright (c) 1988 AT&T */
/*        All Rights Reserved   */


/*
 * Copyright 1999-2003 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * An application should not include this header directly.  Instead it
 * should be included only through the inclusion of other Sun headers.
 *
 * The contents of this header is limited to identifiers specified in the
 * C Standard.  Any new identifiers specified in future amendments to the
 * C Standard must be placed in this header.  If these new identifiers
 * are required to also be in the C++ Standard "std" namespace, then for
 * anything other than macro definitions, corresponding "using" directives
 * must also be added to <stddef.h.h>.
 */

#ifndef _ISO_STDDEF_ISO_H
#define _ISO_STDDEF_ISO_H

#pragma ident   "%Z%%M% %I%     %E% SMI" /* SVr4.0 1.5 */

#include <sys/isa_defs.h>

#ifdef  __cplusplus
extern "C" {
#endif

#if __cplusplus >= 199711L
namespace std {
#endif

#ifndef NULL
#if defined(_LP64)
#define NULL    0L
#else
```



```
#define NULL    0
#endif
#endif

#if !defined(_PTRDIFF_T) || __cplusplus >= 199711L
#define _PTRDIFF_T
#if defined(_LP64) || defined(_I32LPx)
typedef long    ptrdiff_t;              /* pointer difference */
#else
typedef int     ptrdiff_t;              /* (historical version) */
#endif
#endif  /* !_PTRDIFF_T */

#if !defined(_SIZE_T) || __cplusplus >= 199711L
#define _SIZE_T
#if defined(_LP64) || defined(_I32LPx)
typedef unsigned long   size_t;         /* size of something in bytes */
#else
typedef unsigned int    size_t;         /* (historical version) */
#endif
#endif  /* !_SIZE_T */

#if __cplusplus >= 199711L
}
#endif /* end of namespace std */

#if __cplusplus >= 199711L
#define offsetof(s, m)  (std::size_t)(&(((s *)0)->m))
#else
#if defined(__GNUC__)
#define offsetof(s, m)  __builtin_offsetof(s, m)
#else
#define offsetof(s, m)  ((size_t)(&(((s *)0)->m)))
#endif
#define offsetof(s, m)  (size_t)(&(((s *)0)->m))
#endif

#ifdef  __cplusplus
}
_____unchanged_portion_omitted_
```

```
**********************************************************
    3738 Thu Sep  5 23:02:08 2013
new/usr/src/lib/libumem/common/misc.h
3373 gcc >= 4.5 concerns about offsetof()
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _MISC_H
#define _MISC_H

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

#include <sys/types.h>
#include <sys/time.h>
#include <thread.h>
#include <pthread.h>
#include <stdarg.h>

#ifdef  __cplusplus
extern "C" {
#endif

extern uint_t umem_abort;               /* abort when errors occur */
extern uint_t umem_output;              /* output error messages to stderr */
extern caddr_t umem_min_stack;          /* max stack address for audit log */
extern caddr_t umem_max_stack;          /* min stack address for audit log */

/*
 * various utility functions
 * These are globally implemented.
 */

#undef  offsetof
#if defined(__GNUC__)
#define offsetof(s, m)  __builtin_offsetof(s, m)
#else
#endif /* ! codereview */
#define offsetof(s, m)  ((size_t)(&(((s *)0)->m)))
#endif
#endif /* ! codereview */

/*
```



```
 * a safe printf  -- do not use for error messages.
 */
void debug_printf(const char *format, ...);

/*
 * adds a message to the log without writing it out.
 */
void log_message(const char *format, ...);

/*
 * returns the index of the (high/low) bit + 1
 */
int highbit(ulong_t);
int lowbit(ulong_t);
#pragma no_side_effect(highbit, lowbit)

/*
 * Converts a hrtime_t to a timestruc_t
 */
void hrt2ts(hrtime_t hrt, timestruc_t *tsp);

/*
 * tries to print out the symbol and offset of a pointer using umem_error_info
 */
int print_sym(void *pointer);

/*
 * Information about the current error.  Can be called multiple times, should
 * be followed eventually with a call to umem_err or umem_err_recoverable.
 */
void umem_printf(const char *format, ...);
void umem_vprintf(const char *format, va_list);

void umem_printf_warn(void *ignored, const char *format, ...);

void umem_error_enter(const char *);

/*
 * prints error message and stack trace, then aborts.  Cannot return.
 */
void umem_panic(const char *format, ...) __NORETURN;
#pragma does_not_return(umem_panic)
#pragma rarely_called(umem_panic)

/*
 * like umem_err, but only aborts if umem_abort > 0
 */
void umem_err_recoverable(const char *format, ...);

/*
 * We define our own assertion handling since libc's assert() calls malloc()
 */
#ifdef NDEBUG
#define ASSERT(assertion) (void)0
#else
#define ASSERT(assertion) (void)((assertion) || \
    __umem_assert_failed(#assertion, __FILE__, __LINE__))
#endif

int __umem_assert_failed(const char *assertion, const char *file, int line);
#pragma does_not_return(__umem_assert_failed)
#pragma rarely_called(__umem_assert_failed)
/*
 * These have architecture-specific implementations.
 */
```

```
126 /*
127  * Returns the current function's frame pointer.
128  */
129 extern void *getfp(void);

131 /*
132  * puts a pc-only stack trace of up to pcstack_limit frames into pcstack.
133  * Returns the number of stacks written.
134  *
135  * if check_sighandler != 0, and we are in a signal context, calls
136  * umem_err_recoverable.
137  */
138 extern int getpcstack(uintptr_t *pcstack, int pcstack_limit,
139     int check_sighandler);

141 #ifdef  __cplusplus
142 }
143 #endif

145 #endif /* _MISC_H */
```

```
**********************************************************
   49211 Thu Sep  5 23:02:09 2013
new/usr/src/lib/lvm/libmeta/common/meta_statconcise.c
3373 gcc >= 4.5 concerns about offsetof()
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

#include <meta.h>
#include <assert.h>
#include <ctype.h>
#include <mdiox.h>
#include <meta.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <sys/lvm/md_mddb.h>
#include <sys/lvm/md_names.h>
#include <sys/lvm/md_crc.h>
#include <sys/lvm/md_convert.h>


/*
 * Design Notes:
 *
 * All of the code in this file supports the addition of metastat -c output
 * for the verbose option of metaimport.  Some of this code is also used by
 * the command metastat for concise output(cmd/lvm/util/metastat.c).
 * The code is designed to produce the same output as metastat -c does for a
 * given diskset--with a couple exceptions.
 * The primary differences between the output for the metastat -c command and
 * metastat output for metaimport -v are:
 *  - the set name is not printed next to each metadevice
 *  - top-level state information is not printed for some metadevices
 *  - the percent that a disk has completed resyncing is not listed
 * in metaimport -v.
 *
 *
 * The general layout of this file is as follows:
 *
 *  - report_metastat_info()
 *      This is the primary entry point for the functions in this file, with
 *      the exception of several functions that are also called from
 *      cmd/io/lvm/util/metastat.c
 *      report_metastat_info() calls functions to read in all the the
 *      Directory blocks and Record blocks and then process the information
 *      needed to print out the metadevice records in the same format as
 *      metastat -c.
 *
 *  - read_all_mdrecords()
 *      Reads in all the Directory blocks in the diskset and verifies their
 *      validity.  For each Directly block, it loops through all Directory
 *      Entries and for each one that contains a metadevice record calls
 *      read_md_record().  Because the output is designed to imitate the
 *      output of metastat -c, we ignore metadevice records for
 *      optimized resync, changelog, and translog.
 *
 *  - read_md_record()
 *      Reads in a Directory Entry and its associated Record block.  The
 *      revision information for the Record block is checked and it is
 *      determined whether or not it is a 64bit Record block or a 32bit record
 *      block.  For each valid Record block, it allocates an md_im_rec_t
 *      structure and calls extract_mduser_data().
 *
 *  - extract_mduser_data()
 *      Populates the md_im_rec_t data structure with information about the
 *      record's associated metadevice.  Also, the name of the metadevice is
 *      either copied from the NM namespace(if it exists there) or is generated
 *      from the record's un_self_id.
 *
 *  - process_toplevel_devices()
 *      For a given metadevice type, searchs through the md_im_rec_t **mdimpp,
 *      list of all metadevices in the set, to find all records of the
 *      specified type that do not have a parent and puts them on a temp list.
 *      The temp list is then iterated through and the associated processing
 *      function is called.
 *
 *  - process_(trans, hotspare, hotspare_pool, soft_part, mirror, stripe, raid)
 *      These functions are called by using the dfunc field in the mdimpp list.
 *      Each process function only understands its own type of metadevice. Once
 *      it processes the metadevice it was called for, it then loops through
 *      all of the underlying metadevices.  After printing the name of the
 *      underlying metadevice, it puts in on a list to be processed.  If the
 *      underlying device is a physical device, then print_physical_device is
 *      called.
 *      Once all information about the original metadevice is processed, it
 *      loops through the list of underlying metadevices and calls the
 *      appropriate function to process them.
 *
 *  - process_toplevel_softparts()
 *      To match the output for metastat -c, all top-level softpartions
 *      are printed out in groups based on their underlying metadevice--so that
 *      the underlying metadevice only needs to be processed once.
 *
 *  - meta_get_(sm_state, raid_col_state, stripe_state, hs_state)
 *      These functions are used to retrieve the metadevice state information.
 *      They are also used by the metastat concise routines in
 *      cmd/lvm/util/metastat.c.
 *
 */


/*
 * md_im_rec is a doubly linked list used to store the rb_data for each
 * directory entry that corresponds to a metadevice.
 * n_key: is set, if there is an associated entry in the NM namespace.
 * dfunc: is set to point to the function that processes the particular
 * metadevice associated with the record.
```

```
 126  * hs_record_id: is only set, if the metadevice is a hotspare.
 127  * un_self_id: is set for all other records. This is also used to generate
 128  * the name of the metadevice if there is no entry for the metadevice in
 129  * the NM namespace--n_key is not set.
 130  */
 131 typedef struct md_im_rec {
 132         mdkey_t                 n_key; /* NM namespace key */
 133         struct md_im_rec        *next;
 134         struct md_im_rec        *prev;
 135         uint_t                  md_type;
 136         uint_t                  has_parent; /* either 0(no parent) or 1 */
 137         minor_t                 un_self_id;
 138         mddb_recid_t            hs_record_id; /* hotspare recid */
 139         char                    *n_name;  /* name of metadevice */
 140         void                    (*dfunc) ();
 141         ushort_t                record_len;
 142         /* pointer to the unit structure for the metadevice, e.g. rb_data[0] */
 143         void                    *record;
 144 } md_im_rec_t;
_____unchanged_portion_omitted_


 156 /*
 157  * MAXSIZEMDRECNAME is the value that has historically been used to allocate
 158  * space for the metadevice name
 159  */
 160 #define MAXSIZEMDRECNAME        20
 161 #define NAMEWIDTH               16
 162 #if defined(__GNUC__)
 163 #define offsetof(s, m)  __builtin_offsetof(s, m)
 164 #else
 165 #endif /* ! codereview */
 166 #define offsetof(s, m)  ((size_t)(&(((s *)0)->m)))
 167 #endif
 168 #endif /* ! codereview */
 169 #define NOT_PHYSICAL_DEV        0
 170 #define PHYSICAL_DEV            1


 173 /*
 174  * strip_blacks()
 175  *
 176  * Strip blanks from string.  Used for size field in concise output.
 177  */
 178 static char *
 179 strip_blanks(char *s)
 180 {
 181         char *p;

 183         for (p = s; *p; ) {
 184                 if (*p == ' ') {
 185                         char *t;
 186                         for (t = p; *t; t++) {
 187                                 *t = *(t + 1);
 188                         }
 189                 } else {
 190                         p++;
 191                 }
 192         }

 194         return (s);
 195 }


 198 /*
 199  * print_concise_entry()
```



```
 200  *
 201  * Print properly indented metadevice name, type and size for concise output.
 202  * This function is also called from: cmd/lvm/util/metastat.c.
 203  */
 204 void
 205 print_concise_entry(int indent, char *name, diskaddr_t size, char mtype)
 206 {
 207         int     i;
 208         int     width = NAMEWIDTH;      /* minumum field width for name */
 209         char    in[MAXPATHLEN];
 210         char    *sz;

 212         in[0] = 0;
 213         for (i = 0; i < indent; i++)
 214                 (void) strlcat(in, " ", sizeof (in));

 216         /* set up minimum field width. negative for left justified */
 217         width -= indent;
 218         if (width < 0)
 219                 width = 0;      /* overflowed; no minimum field needed */
 220         else
 221                 width = 0 - width; /* negative for left justification */

 223         if (size == 0) {
 224                 sz = "-";
 225         } else {
 226                 sz = strip_blanks(meta_number_to_string(size, DEV_BSIZE));
 227         }

 229         (void) printf("%s%*s %c %6s", in, width, name, mtype, sz);
 230 }


 233 /*
 234  * free_mdrec_list_entry()
 235  *
 236  * Removing entry from the list of metadevices in the diskset(mdimpp).
 237  * This function will not remove the dummy entry at the head of the
 238  * list, so we don't have to set mdrec equal to NULL.
 239  */
 240 static void
 241 free_mdrec_list_entry(md_im_rec_t  **mdrec)
 242 {
 243         (*mdrec)->prev->next = (*mdrec)->next;
 244         if ((*mdrec)->next != NULL) {
 245                 (*mdrec)->next->prev = (*mdrec)->prev;
 246         }
 247         Free((*mdrec)->record);
 248         Free((*mdrec)->n_name);
 249         Free(*mdrec);
 250 }


 253 /*
 254  * ucomponent_append()
 255  *
 256  * Appending entry to the underlying component list.  The list
 257  * is used to group all of the underlying devices before
 258  * processing them.
 259  */
 260 static void
 261 ucomponent_append(
 262         md_im_list_t    **ucomp_head,
 263         md_im_list_t    **ucomp_tail,
 264         md_im_list_t    *ucomp
 265 )
```

```
{
        ucomp->next = NULL;
        if (*ucomp_head == NULL) {
                *ucomp_head = ucomp;
                *ucomp_tail = ucomp;
        } else {
                (*ucomp_tail)->next = ucomp;
                *ucomp_tail = (*ucomp_tail)->next;
        }
}


/*
 * free_md_im_list_entries()
 *
 * Freeing entries on an md_im_list_t.  This list is used to group
 * underlying components for processing and to group top-level metadevices
 * by type.
 */
static void
free_md_im_list_entries(md_im_list_t **list_head)
{
        md_im_list_t    *tmp_list_entry = *list_head;
        md_im_list_t    *rm_list_entry;

        while (tmp_list_entry != NULL) {
                rm_list_entry = tmp_list_entry;
                tmp_list_entry = tmp_list_entry->next;
                Free(rm_list_entry);
        }
}


/*
 * print_physical_device()
 *
 * If a metadevice has an underlying component that is a physical
 * device, then this searches the pnm_rec_t list to match an entry's
 * n_key to the key for the underlying component.  The ctd name of the
 * physical device is printed on the same line as the metadevice.
 */
static void
print_physical_device(
        pnm_rec_t       *phys_nm,
        mdkey_t         key
)
{
        pnm_rec_t       *tmpphys_nm;

        for (tmpphys_nm = phys_nm; tmpphys_nm != NULL;
            tmpphys_nm = tmpphys_nm->next) {
                if (tmpphys_nm->n_key == key) {
                        (void) printf(" %s", tmpphys_nm->n_name);
                        break;
                }
        }
}


/*
 * get_stripe_req_size()
 *
 * Given a 64bit stripe unit, compute the size of the stripe unit.
 * This function is a derivation of:
 *      common/lvm/md_convert.c:get_big_stripe_req_size()
 * and any changes made to either this function or get_big_stripe_req_size()
```



```
 * should be reviewed to make sure the functionality in both places is correct.
 *
 * Returns:
 *      total size of the 64bit stripe
 */
size_t
get_stripe_req_size(ms_unit_t *un)
{
        struct ms_row *mdr;
        uint_t row;
        uint_t ncomps = 0;
        size_t mdsize = 0;
        size_t first_comp = 0;


        /* Compute the offset of the first component */
        first_comp = sizeof (ms_unit_t) +
            sizeof (struct ms_row) * (un->un_nrows - 1);
        first_comp = roundup(first_comp, sizeof (long long));

        /*
         * Requestor wants to have the total size, add the sizes of
         * all components
         */
        mdr = &un->un_row[0];
        for (row = 0; (row < un->un_nrows); row++)
            ncomps += mdr[row].un_ncomp;
        mdsize = first_comp + sizeof (ms_comp_t) * ncomps;
        return (mdsize);
}


/*
 * meta_get_sm_state()
 *
 * Gets the state for the underlying components(submirrors) of a mirror.
 * This function is also called from: cmd/lvm/util/metastat.c.
 *
 * Returns:
 *      string for state of the sub-mirror
 */
static char *
meta_get_sm_state(
        sm_state_t      state
)
{
        /* all is well */
        if (state & SMS_RUNNING) {
                return (NULL);
        }

        /* resyncing, needs repair */
        if ((state & (SMS_COMP_RESYNC | SMS_ATTACHED_RESYNC |
            SMS_OFFLINE_RESYNC))) {
                return (gettext("resyncing"));
        }

        /* needs repair */
        if (state & (SMS_COMP_ERRED | SMS_ATTACHED | SMS_OFFLINE))
                return (gettext("maint"));

        /* unknown */
        return (gettext("unknown"));
}
```

```
/*
 * meta_get_raid_col_state()
 *
 * Gets the state for the underlying components(columns) of a raid.
 * This function is also called from: cmd/lvm/util/metastat.c.
 *
 * Returns:
 *      string for state of the raid column
 *
 */
char *
meta_get_raid_col_state(
        rcs_state_t     state
)
{
        switch (state) {
                case RCS_INIT:
                        return (gettext("initializing"));
                case RCS_OKAY:
                        return (NULL);
                case RCS_INIT_ERRED:
                        /*FALLTHROUGH*/
                case RCS_ERRED:
                        return (gettext("maint"));
                case RCS_LAST_ERRED:
                        return (gettext("last-erred"));
                case RCS_RESYNC:
                        return (gettext("resyncing"));
                default:
                        return (gettext("unknown"));
        }
}


/*
 * meta_get_stripe_state()
 *
 * Gets the state for the underlying components of a stripe.
 * This function is also called from: cmd/lvm/util/metastat.c.
 *
 * Returns:
 *      string for state of the stripe
 *
 */
char *
meta_get_stripe_state(
        comp_state_t    state
)
{
        switch (state) {
                case CS_OKAY:
                        return (NULL);
                case CS_ERRED:
                        return (gettext("maint"));
                case CS_LAST_ERRED:
                        return (gettext("last-erred"));
                case CS_RESYNC:
                        return (gettext("resyncing"));
                default:
                        return (gettext("invalid"));
        }
}


/*
 * meta_get_hs_state()
```



```
 *
 * Gets the state for the underlying components(hotspares) of a hotspare pool.
 * This function is also called from: cmd/lvm/util/metastat.c.
 *
 * Returns:
 *      string for state of the hotspare
 *
 */
char *
meta_get_hs_state(
        hotspare_states_t       state
)
{
        switch (state) {
                case HSS_AVAILABLE:
                        return (NULL);
                case HSS_RESERVED:
                        return (gettext("in-use"));
                case HSS_BROKEN:
                        return (gettext("broken"));
                case HSS_UNUSED:
                        /* FALLTHROUGH */
                default:
                        return (gettext("invalid"));
        }
}


/*
 * process_trans()
 *
 * Prints unit information for a trans metadevice and calls the respective
 * functions to process the underlying metadevices.
 *
 */
static void
process_trans(
        md_im_rec_t     **mdimpp,
        int             indent,
        pnm_rec_t       *phys_nm,
        md_im_rec_t     *mdrec
)
{
        mt_unit_t       *mt;
        mdc_unit_t      uc;
        md_im_rec_t     *tmpmdrec;
        int             underlying_device = PHYSICAL_DEV;

        mt = (mt_unit_t *)mdrec->record;
        uc = mt->c;

        /* Printing name, size, and type of metadevice */
        print_concise_entry(indent, mdrec->n_name,
            uc.un_total_blocks, 't');

        /*
         * Loops through md_im_rec_t **mdimpp list of all metadevices to find
         * record that matches the underlying device.
         * Trans devices can only have one underlying device, so once a
         * match is found, we are done.
         */
        for (tmpmdrec = *mdimpp; tmpmdrec != NULL;
            tmpmdrec = tmpmdrec->next) {
                if (tmpmdrec->n_key == mt->un_m_key) {
                        /* Printing name of the underlying metadevice */
                        (void) printf(" %s", tmpmdrec->n_name);
```

```
530                         underlying_device = NOT_PHYSICAL_DEV;
531                         break;
532                 }
533         }

535         /*
536          * If a metadevice was not found, then the underlying device must be a
537          * physical device.  Otherwise, call the functions to process the
538          * underlying devices.
539          */
540         if (underlying_device == PHYSICAL_DEV) {
541                 print_physical_device(phys_nm, mt->un_m_key);
542                 (void) printf("\n");
543         } else {
544                 /* process underlying component */
545                 (void) printf("\n");
546                 indent += META_INDENT;
547                 tmpmdrec->dfunc(mdimpp, indent, phys_nm, tmpmdrec);
548         }

550         /*
551          * Removing the md_entry from the list
552          * of all metadevices
553          */
554         free_mdrec_list_entry(&mdrec);
555 }


558 /*
559  * process_hotspare()
560  *
561  * Searches though list of physical devices to match hotspare record.
562  * Prints physical device name and state of a hotspare unit.
563  *
564  */
565 /*ARGSUSED*/
566 static void
567 process_hotspare(
568         md_im_rec_t     **mdimpp,
569         int             indent,
570         pnm_rec_t       *phys_nm,
571         md_im_rec_t     *mdrec
572 )
573 {
574         hot_spare_t     *hs;
575         pnm_rec_t       *tmpphys_nm;
576         char            *state = NULL;

578         hs =  (hot_spare_t *)mdrec->record;

580         /*
581          * Loops through physical namespace to find the device that matches
582          * the hotspare entry.
583          */
584         for (tmpphys_nm = phys_nm; tmpphys_nm != NULL;
585             tmpphys_nm = tmpphys_nm->next) {
586                 if (tmpphys_nm->n_key ==
587                     ((hot_spare_t *)hs)->hs_key) {
588                         /* Printing name of hotspare device */
589                         (void) printf(" %s", tmpphys_nm->n_name);
590                         break;
591                 }
592         }

594         state = meta_get_hs_state(hs->hs_state);
595         if (state != NULL)
```



```
596                 (void) printf(" (%s)", state);

598         /* Not removing entry, because it can be processed more than once. */
599 }


602 /*
603  * process_hotspare_pool()
604  *
605  * Prints concise unit information for a hotspare pool metadevice and calls a
606  * function to process each attached hotspare device.
607  *
608  */
609 static void
610 process_hotspare_pool(
611         md_im_rec_t     **mdimpp,
612         int             indent,
613         pnm_rec_t       *phys_nm,
614         md_im_rec_t     *mdrec
615 )
616 {
617         hot_spare_pool_ond_t    *hsp;
618         int                     i;
619         md_im_rec_t             *tmpmdrec;

621         hsp =  (hot_spare_pool_ond_t *)mdrec->record;

623         /*
624          * Printing name, size, and type of metadevice. Setting size field to
625          * 0, so that output is the as metastat -c.
626          */
627         print_concise_entry(indent, mdrec->n_name,
628             0, 'h');

630         /* Looping through list of attached hotspare devices. */
631         for (i = 0; i < hsp->hsp_nhotspares; i++) {
632                 /* Looking for the matching record for the hotspare device. */
633                 for (tmpmdrec = *mdimpp; tmpmdrec != NULL;
634                     tmpmdrec = tmpmdrec->next) {
635                         if (tmpmdrec->hs_record_id == hsp->hsp_hotspares[i]) {
636                                 /* Calling function to print name of hotspare */
637                                 tmpmdrec->dfunc(mdimpp, indent, phys_nm,
638                                     tmpmdrec);
639                         }
640                 }
641         }
642         (void) printf("\n");

644         /*
645          * Removing the md_entry from the list
646          * of all metadevices
647          */
648         free_mdrec_list_entry(&mdrec);
649 }


652 /*
653  * process_raid()
654  *
655  * Prints concise unit information for a raid metadevice and calls the
656  * respective functions to process the underlying metadevices.
657  *
658  */
659 static void
660 process_raid(
661         md_im_rec_t     **mdimpp,
```

```
662         int             indent,
663         pnm_rec_t       *phys_nm,
664         md_im_rec_t     *mdrec
665 )
666 {
667         mr_unit_t       *mr;
668         mr_column_t     *mc;
669         mdc_unit_t      uc;
670         int             i;
671         md_im_rec_t     *tmpmdrec;
672         md_im_rec_t     *hstmpmdrec;
673         md_im_list_t    *ucomp_head = NULL;
674         md_im_list_t    *ucomp_tail = NULL;
675         md_im_list_t    *ucomp = NULL;
676         pnm_rec_t       *tmpphys_nm;
677         int             underlying_device;

679         mr =  (mr_unit_t *)mdrec->record;
680         uc = mr->c;

682         /* Printing name, size, and type of metadevice */
683         print_concise_entry(indent, mdrec->n_name,
684             uc.un_total_blocks, 'r');

686         /* Loops through raid columns to find underlying metadevices */
687         for (i = 0, mc = &mr->un_column[0];  i < mr->un_totalcolumncnt;
688             i++, mc++) {
689                 char    *state = NULL;
690                 char    *hsname = NULL;

692                 /*
693                  * Need to assume that underlying device is a physical device,
694                  * unless we find a matching metadevice record.
695                  */
696                 underlying_device = PHYSICAL_DEV;

698                 /*
699                  * Loops through list of metadevices to find record that matches
700                  * the underlying device.
701                  */
702                 for (tmpmdrec = *mdimpp; tmpmdrec != NULL;
703                     tmpmdrec = tmpmdrec->next) {
704                         if (tmpmdrec->n_key == mc->un_orig_key) {
705                                 /* check if hotspare device enabled */
706                                 if (mc->un_hs_id !=  NULL) {
707                                         /*
708                                          * Find matching metadevice record
709                                          * for the hotspare device.
710                                          */
711                                         for (hstmpmdrec = *mdimpp;
712                                             hstmpmdrec != NULL;
713                                             hstmpmdrec = hstmpmdrec->next) {
714                                                 if (hstmpmdrec->hs_record_id ==
715                                                     mc->un_hs_id) {
716                                                         /* print name of hs */
717                                                         hstmpmdrec->dfunc(
718                                                             mdimpp, indent,
719                                                             phys_nm,
720                                                             hstmpmdrec);
721                                                         break;
722                                                 }
723                                         }
724                                 }
725                                 /* print name of underlying metadevice */
726                                 (void) printf(" %s", tmpmdrec->n_name);
727                                 underlying_device = NOT_PHYSICAL_DEV;
```



```
728                                 ucomp = Zalloc(sizeof (md_im_list_t));
729                                 ucomp->mdrec = tmpmdrec;
730                                 ucomponent_append(&ucomp_head, &ucomp_tail,
731                                     ucomp);
732                         }
733                 }

735                 if (underlying_device == PHYSICAL_DEV) {
736                         print_physical_device(phys_nm, mc->un_orig_key);
737                 }
738                 state = meta_get_raid_col_state(mc->un_devstate);

740                 /*
741                  * An underlying hotspare must be a physical device.
742                  * If support is ever added for soft-partitions under
743                  * hotspare pools, then this code should be updated to
744                  * include a search for underlying metadevices.
745                  */
746                 if (mc->un_hs_id != 0) {
747                         for (tmpphys_nm = phys_nm; tmpphys_nm != NULL;
748                             tmpphys_nm = tmpphys_nm->next) {
749                                 if (tmpphys_nm->n_key == mc->un_hs_key) {
750                                         hsname = tmpphys_nm->n_name;
751                                         break;
752                                 }
753                         }
754                 }

756                 if (state != NULL) {
757                         if (hsname != NULL)
758                                 (void) printf(" (%s-%s)", state,
759                                     hsname);
760                         else
761                                 (void) printf(" (%s)", state);
762                 } else if (hsname != NULL) {
763                         (void) printf(gettext(" (spared-%s)"), hsname);
764                 }
765         }
766         (void) printf("\n");

768         /* process underlying components */
769         indent += META_INDENT;
770         for (ucomp = ucomp_head; ucomp != NULL;
771             ucomp = ucomp->next) {
772                 ucomp->mdrec->dfunc(mdimpp, indent, phys_nm,
773                     ucomp->mdrec);
774         }
775         free_md_im_list_entries(&ucomp_head);

777         /*
778          * Removing the md_entry from the list
779          * of all metadevices
780          */
781         free_mdrec_list_entry(&mdrec);
782 }


785 /*
786  * process_mirror()
787  *
788  * Prints concise unit information for a mirror metadevice and calls the
789  * respective functions to process the underlying metadevices.
790  *
791  */
792 static void
793 process_mirror(
```

```
794         md_im_rec_t     **mdimpp,
795         int             indent,
796         pnm_rec_t       *phys_nm,
797         md_im_rec_t     *mdrec
798 )
799 {
800         mm_unit_t       *mm;
801         mm_submirror_t  *sm;
802         mdc_unit_t      uc;
803         int             i;
804         md_im_rec_t     *tmpmdrec;
805         md_im_list_t    *ucomp_head = NULL;
806         md_im_list_t    *ucomp_tail = NULL;
807         md_im_list_t    *ucomp = NULL;

809         mm =  (mm_unit_t *)mdrec->record;
810         uc = mm->c;

812         /* Printing name, size, and type of metadevice */
813         print_concise_entry(indent, mdrec->n_name,
814             uc.un_total_blocks, 'm');

816         /* Looping through sub-mirrors to find underlying devices */
817         for (i = 0, sm = &mm->un_sm[0]; i < mm->un_nsm; i++, sm++) {
818                 char    *state = NULL;

820                 for (tmpmdrec = *mdimpp; tmpmdrec != NULL;
821                     tmpmdrec = tmpmdrec->next) {
822                         if (tmpmdrec->n_key == sm->sm_key) {
823                                 (void) printf(" %s", tmpmdrec->n_name);
824                                 ucomp = Zalloc(sizeof (md_im_list_t));
825                                 ucomp->mdrec = tmpmdrec;
826                                 ucomponent_append(&ucomp_head, &ucomp_tail,
827                                     ucomp);
828                         }
829                 }

831                 /*
832                  * It is not possible to have an underlying physical device
833                  * for a submirror, so there is no need to search the phys_nm
834                  * list.
835                  */

837                 /* Printing the state for the submirror */
838                 state = meta_get_sm_state(sm->sm_state);
839                 if (state != NULL) {
840                         (void) printf(" (%s)", state);
841                 }
842         }
843         (void) printf("\n");

845         /* process underlying components */
846         indent += META_INDENT;
847         for (ucomp = ucomp_head; ucomp != NULL;
848             ucomp = ucomp->next) {
849                 ucomp->mdrec->dfunc(mdimpp, indent, phys_nm,
850                     ucomp->mdrec);
851         }
852         free_md_im_list_entries(&ucomp_head);

854         /*
855          * Removing the md_entry from the list
856          * of all metadevices
857          */
858         free_mdrec_list_entry(&mdrec);
859 }
```



```
862 /*
863  * process_stripe()
864  *
865  * Prints concise unit information for a stripe metadevice and calls the
866  * respective functions to process the underlying metadevices.
867  *
868  */
869 static void
870 process_stripe(
871         md_im_rec_t     **mdimpp,
872         int             indent,
873         pnm_rec_t       *phys_nm,
874         md_im_rec_t     *mdrec
875 )
876 {
877         ms_unit_t       *ms;
878         mdc_unit_t      uc;
879         md_im_rec_t     *tmpmdrec;
880         md_im_list_t    *ucomp_head = NULL;
881         md_im_list_t    *ucomp_tail = NULL;
882         md_im_list_t    *ucomp = NULL;
883         pnm_rec_t       *tmpphys_nm;
884         int             underlying_device;
885         uint_t          row;

887         ms =  (ms_unit_t *)mdrec->record;
888         uc = ms->c;

890         /* Printing name, size, and type of metadevice */
891         print_concise_entry(indent, mdrec->n_name,
892             uc.un_total_blocks, 's');

894         /* Looping through stripe rows */
895         for (row = 0; (row < ms->un_nrows); ++row) {
896                 struct ms_row   *mdr = &ms->un_row[row];
897                 ms_comp_t       *mdcomp = (void *)&((char *)ms)
898                     [ms->un_ocomp];
899                 uint_t          comp, c;

901                 /*
902                  * Looping through the components in each row to find the
903                  * underlying devices.
904                  */
905                 for (comp = 0, c = mdr->un_icomp; (comp < mdr->un_ncomp);
906                     ++comp, ++c) {
907                         char            *state = NULL;
908                         char            *hsname = NULL;
909                         ms_comp_t       *mdc = &mdcomp[c];
910                         md_m_shared_t   *mdm = &mdc->un_mirror;

912                         /*
913                          * Need to assume that underlying device is a
914                          * physical device, unless we find a matching
915                          * metadevice record.
916                          */
917                         underlying_device = PHYSICAL_DEV;

919                         for (tmpmdrec = *mdimpp; tmpmdrec != NULL;
920                             tmpmdrec = tmpmdrec->next) {
921                                 if (tmpmdrec->n_key == mdc->un_key) {
922                                         (void) printf(" %s", tmpmdrec->n_name);
923                                         underlying_device = NOT_PHYSICAL_DEV;
924                                         ucomp = Zalloc(sizeof (md_im_list_t));
925                                         ucomp->mdrec = tmpmdrec;
```

```
926                                     ucomponent_append(&ucomp_head,
927                                         &ucomp_tail, ucomp);
928                             }
929                         }
930                         /* if an underlying metadevice was not found */
931                         if (underlying_device == PHYSICAL_DEV) {
932                                 print_physical_device(phys_nm, mdc->un_key);
933                         }
934                         state = meta_get_stripe_state(mdm->ms_state);

936                         /*
937                          * An underlying hotspare must be a physical device.
938                          * If support is ever added for soft-partitions under
939                          * hotspare pools, then this code should be updated to
940                          * include a search for underlying metadevices.
941                          */
942                         if (mdm->ms_hs_key != 0) {
943                                 for (tmpphys_nm = phys_nm; tmpphys_nm != NULL;
944                                     tmpphys_nm = tmpphys_nm->next) {
945                                         if (tmpphys_nm->n_key ==
946                                             mdm->ms_hs_key) {
947                                                 hsname = tmpphys_nm->n_name;
948                                                 break;
949                                         }
950                                 }
951                         }
952                         if (state != NULL) {
953                                 if (hsname != NULL) {
954                                         (void) printf(" (%s-%s)", state,
955                                             hsname);
956                                 } else {
957                                         (void) printf(" (%s)", state);
958                                 }
959                         } else if (hsname != NULL) {
960                                 (void) printf(gettext(" (spared-%s)"), hsname);
961                         }
962                 }
963         }
964         (void) printf("\n");

966         /* Process underlying metadevices */
967         indent += META_INDENT;
968         for (ucomp = ucomp_head; ucomp != NULL;
969             ucomp = ucomp->next) {
970                 ucomp->mdrec->dfunc(mdimpp, indent, phys_nm,
971                     ucomp->mdrec);
972         }
973         free_md_im_list_entries(&ucomp_head);

975         /*
976          * Removing the md_entry from the list
977          * of all metadevices
978          */
979         free_mdrec_list_entry(&mdrec);
980 }


983 /*
984  * process_softpart()
985  *
986  * Prints concise unit information for a softpart metadevice and calls the
987  * respective functions to process the underlying metadevices.
988  *
989  */
990 static void
991 process_softpart(
```



```
992         md_im_rec_t     **mdimpp,
993         int             indent,
994         pnm_rec_t       *phys_nm,
995         md_im_rec_t     *mdrec
996 )
997 {
998         mp_unit_t       *mp;
999         mdc_unit_t      uc;
1000        md_im_rec_t     *tmpmdrec;
1001        int             underlying_device = PHYSICAL_DEV;

1003        mp =  (mp_unit_t *)mdrec->record;
1004        uc = mp->c;

1006        /* Printing name, size, and type of metadevice */
1007        print_concise_entry(indent, mdrec->n_name,
1008            uc.un_total_blocks, 'p');

1010        /*
1011         * Loops through md_im_rec_t **mdimpp list of all metadevices to find
1012         * record that matches the underlying device.
1013         * Softpartitions can only have one underlying device, so once a
1014         * match is found, we are done.
1015         */
1016        for (tmpmdrec = *mdimpp; tmpmdrec != NULL;
1017            tmpmdrec = tmpmdrec->next) {
1018                if (tmpmdrec->n_key == mp->un_key) {
1019                        /* Printing name of the underlying metadevice */
1020                        (void) printf(" %s", tmpmdrec->n_name);
1021                        underlying_device = NOT_PHYSICAL_DEV;
1022                        break;
1023                }
1024        }

1026        /* This is only executed if an underlying metadevice was not found */
1027        if (underlying_device == PHYSICAL_DEV) {
1028                print_physical_device(phys_nm, mp->un_key);
1029                (void) printf("\n");
1030        } else {
1031                /* Process underlying metadevice */
1032                (void) printf("\n");
1033                indent += META_INDENT;
1034                tmpmdrec->dfunc(mdimpp, indent, phys_nm,
1035                    tmpmdrec);
1036        }

1038        /*
1039         * Removing the md_entry from the list
1040         * of all metadevices
1041         */
1042        free_mdrec_list_entry(&mdrec);
1043 }


1046 /*
1047  * process_toplevel_softparts()
1048  *
1049  * Toplevel softpartions need to be grouped so that their underlying devices
1050  * can be printed just once.
1051  */
1052 static void
1053 process_toplevel_softparts(
1054         md_im_rec_t     **mdimpp,
1055         int             indent,
1056         pnm_rec_t       *phys_nm
1057 )
```

```
1058 {
1059         mp_unit_t       *mp;
1060         mdc_unit_t      uc;
1061         md_im_rec_t     *mdrec;
1062         md_im_rec_t     *comp_mdrec; /* pntr to underlying component's record */
1063         md_im_rec_t     *tmp_mdrec, *rm_mdrec;
1064         mp_unit_t       *tmp_mp;
1065         int             underlying_device;

1067         /*
1068          * Loops through md_im_rec_t **mdimpp list of all metadevices to find
1069          * all softpartions that are toplevel softpartitions(softparts w/out
1070          * a parent). Groups output for these entries so that the function to
1071          * process the underlying metadevice is only called once.
1072          */
1073         for (mdrec = *mdimpp; mdrec != NULL; mdrec = mdrec->next) {

1075                 underlying_device = PHYSICAL_DEV;
1076                 if ((mdrec->md_type == MDDB_F_SOFTPART) &&
1077                     (mdrec->has_parent == 0)) {
1078                         mp =  (mp_unit_t *)mdrec->record;
1079                         uc = mp->c;
1080                         /* Printing name, size, and type of metadevice */
1081                         print_concise_entry(indent, mdrec->n_name,
1082                             uc.un_total_blocks, 'p');
1083                         /*
1084                          * Looking for record that matches underlying
1085                          * component.
1086                          */
1087                         for (comp_mdrec = *mdimpp; comp_mdrec != NULL;
1088                             comp_mdrec = comp_mdrec->next) {
1089                                 if (comp_mdrec->n_key == mp->un_key) {
1090                                         /* Print name of underlying device */
1091                                         (void) printf(" %s",
1092                                             comp_mdrec->n_name);
1093                                         underlying_device = NOT_PHYSICAL_DEV;
1094                                         break;
1095                                 }
1096                         }
1097                         if (underlying_device == PHYSICAL_DEV) {
1098                                 print_physical_device(phys_nm, mp->un_key);
1099                         }
1100                         (void) printf("\n");

1102                         /*
1103                          * Looking for any other toplevel softpartitions with
1104                          * same underlying device. We know that all other
1105                          * matching metadevices, that share the same underlying
1106                          * metadevice, are also soft-partitions.
1107                          */
1108                         for (tmp_mdrec = mdrec->next; tmp_mdrec != NULL; ) {
1109                                 tmp_mp = (mp_unit_t *)tmp_mdrec->record;
1110                                 if ((tmp_mdrec->has_parent == 0) &&
1111                                     (tmp_mp->un_key == mp->un_key)) {
1112                                         uc = tmp_mp->c;
1113                                         print_concise_entry(indent,
1114                                             tmp_mdrec->n_name,
1115                                             uc.un_total_blocks, 'p');
1116                                         if (underlying_device ==
1117                                             NOT_PHYSICAL_DEV) {
1118                                                 (void) printf(" %s",
1119                                                     comp_mdrec->n_name);
1120                                         } else {
1121                                                 print_physical_device(
1122                                                     phys_nm, tmp_mp->un_key);
1123                                         }
```



```
1124                                         (void) printf("\n");
1125                                         /*
1126                                          * Need to advance so that will not lose
1127                                          * position after removing processed
1128                                          * record.
1129                                          */
1130                                         rm_mdrec = tmp_mdrec;
1131                                         tmp_mdrec = tmp_mdrec->next;
1132                                         /*
1133                                          * Removing the md_entry from the list
1134                                          * of all metadevices.
1135                                          */
1136                                         free_mdrec_list_entry(&rm_mdrec);
1137                                 } else {
1138                                         tmp_mdrec = tmp_mdrec->next;
1139                                 }
1140                         }
1141                         /* Process the underlying device */
1142                         if (underlying_device == NOT_PHYSICAL_DEV) {
1143                                 indent += META_INDENT;
1144                                 comp_mdrec->dfunc(mdimpp, indent, phys_nm,
1145                                     comp_mdrec);
1146                         }
1147                 }
1148         }
1149 }


1152 /*
1153  * process_toplevel_devices()
1154  *
1155  * Search through list of metadevices for metadevices of md_type that do not
1156  * have a parent.
1157  *
1158  */
1159 static void
1160 process_toplevel_devices(
1161         md_im_rec_t     **mdimpp,
1162         pnm_rec_t       *pnm,
1163         uint_t          md_type
1164 )
1165 {
1166         md_im_rec_t     *mdrec;
1167         md_im_list_t    *mdrec_tl_tail = NULL;
1168         md_im_list_t    *mdrec_tl_head = NULL;
1169         md_im_list_t    *tmp_tl_list = NULL;
1170         int             indent = 0;

1172         indent += META_INDENT;

1174         /*
1175          * Need to group soft partitions so that common underlying device
1176          * are only processed once.
1177          */
1178         if (md_type == MDDB_F_SOFTPART) {
1179                 process_toplevel_softparts(mdimpp, indent, pnm);
1180                 return;
1181         }

1183         /*
1184          * Search the list of metadevices to find all metadevices that match
1185          * the type and don't have a parent.  Put them on a separate list
1186          * that will be processed.
1187          */
1188         for (mdrec = *mdimpp; mdrec != NULL; mdrec = mdrec->next) {
1189                 if ((mdrec->md_type == md_type)&&(mdrec->has_parent == 0)) {
```

```
1190                        tmp_tl_list = Zalloc(sizeof (md_im_list_t));
1191                        tmp_tl_list->mdrec = mdrec;
1192                        tmp_tl_list->next = NULL;
1193                        if (mdrec_tl_tail == NULL) {
1194                                mdrec_tl_tail = tmp_tl_list;
1195                                mdrec_tl_head = mdrec_tl_tail;
1196                        } else {
1197                                mdrec_tl_tail->next = tmp_tl_list;
1198                                mdrec_tl_tail = mdrec_tl_tail->next;
1199                        }
1200                }

1202        }

1204        /*
1205         * Loop through list and process all top-level metadevices of a
1206         * given type.
1207         */
1208        for (tmp_tl_list = mdrec_tl_head; tmp_tl_list != NULL;
1209            tmp_tl_list = tmp_tl_list->next) {
1210                tmp_tl_list->mdrec->dfunc(mdimpp, indent, pnm,
1211                    tmp_tl_list->mdrec);
1212        }

1214        free_md_im_list_entries(&mdrec_tl_head);
1215 }


1218 /*
1219  * extract_mduser_data()
1220  *
1221  * Converts or copies the (mddb_rb_t->rb_data) metadevice record to a 64bit
1222  * record.
1223  * Sets the dfunc field to point to the appropriate function to process the
1224  * metadevice.
1225  * Sets the parent field for the metadevice.
1226  * Extracts the name from the NM namespace if it is available, otherwise
1227  * generates it from the metadevice's minor number.
1228  *
1229  * Returns:
1230  *      < 0 for failure
1231  *        0 for success
1232  *
1233  */
1234 static int
1235 extract_mduser_data(
1236        mddb_rb_t               *nm,
1237        md_im_rec_t             *mdrec,
1238        void                    *rbp,
1239        int                     is_32bit_record,
1240        md_error_t              *ep
1241 )
1242 {
1243        mdc_unit_t              uc;
1244        hot_spare_t             *hs;
1245        hot_spare_pool_ond_t    *hsp;
1246        size_t                  newreqsize;
1247        mddb_rb_t               *rbp_nm = nm;
1248        struct nm_rec           *nm_record;
1249        struct nm_name          *nmname;
1250        char                    *uname = NULL;


1253        /*LINTED*/
1254        nm_record = (struct nm_rec *)((caddr_t)(&rbp_nm->rb_data));
```



```
1256        /*
1257         * Setting the un_self_id or the hs_self_id, in the case of hotspare
1258         * records, for each metadevice entry. Also setting has_parent and
1259         * setting dfunc so that it points to the correct function to process
1260         * the record type.
1261         * If the record was stored ondisk in 32bit format, then it is
1262         * converted to the 64bits equivalent 64bit format and the memory
1263         * for the 32bit pointer is freed.
1264         */
1265        switch (mdrec->md_type) {
1266                case MDDB_F_SOFTPART:
1267                        if (is_32bit_record) {
1268                                mp_unit32_od_t  *small_un;
1269                                mp_unit_t       *big_un;

1271                                small_un = (mp_unit32_od_t *)((uintptr_t)rbp +
1272                                    (sizeof (mddb_rb_t) - sizeof (int)));
1273                                newreqsize = sizeof (mp_unit_t) +
1274                                    ((small_un->un_numexts - 1) *
1275                                    sizeof (struct mp_ext));
1276                                big_un = (void *)Zalloc(newreqsize);
1277                                softpart_convert((caddr_t)small_un,
1278                                    (caddr_t)big_un, SMALL_2_BIG);
1279                                mdrec->record = (void *)big_un;
1280                        } else {
1281                                mp_unit_t       *big_un;

1283                                big_un = (mp_unit_t *)((uintptr_t)rbp +
1284                                    (sizeof (mddb_rb_t) - sizeof (int)));
1285                                newreqsize = sizeof (mp_unit_t) +
1286                                    ((big_un->un_numexts - 1) *
1287                                    sizeof (struct mp_ext));
1288                                mdrec->record = (void *)Zalloc(newreqsize);
1289                                bcopy(big_un, mdrec->record, newreqsize);
1290                        }
1291                        uc = ((mp_unit_t *)mdrec->record)->c;
1292                        mdrec->dfunc = &process_softpart;
1293                        mdrec->un_self_id = uc.un_self_id;
1294                        mdrec->has_parent = MD_HAS_PARENT(
1295                            uc.un_parent);
1296                        break;
1297                case MDDB_F_STRIPE:
1298                        if (is_32bit_record) {
1299                                ms_unit32_od_t  *small_un;
1300                                ms_unit_t       *big_un;

1302                                small_un = (ms_unit32_od_t *)((uintptr_t)rbp +
1303                                    (sizeof (mddb_rb_t) - sizeof (int)));
1304                                newreqsize = get_big_stripe_req_size(
1305                                    small_un, COMPLETE_STRUCTURE);
1306                                    big_un = (void *)Zalloc(newreqsize);
1307                                stripe_convert((caddr_t)small_un,
1308                                    (caddr_t)big_un, SMALL_2_BIG);
1309                                mdrec->record = (void *)big_un;
1310                        } else {
1311                                ms_unit_t       *big_un;

1313                                big_un = (ms_unit_t *)((uintptr_t)rbp +
1314                                    (sizeof (mddb_rb_t) - sizeof (int)));
1315                                newreqsize = get_stripe_req_size(big_un);
1316                                mdrec->record = (void *)Zalloc(newreqsize);
1317                                bcopy(big_un, mdrec->record, newreqsize);
1318                        }
1319                        uc = ((ms_unit_t *)mdrec->record)->c;
1320                        mdrec->dfunc = &process_stripe;
1321                        mdrec->un_self_id = uc.un_self_id;
```

```
1322                        mdrec->has_parent = MD_HAS_PARENT(
1323                            uc.un_parent);
1324                        break;
1325                case MDDB_F_MIRROR:
1326                        if (is_32bit_record) {
1327                                mm_unit32_od_t  *small_un;
1328                                mm_unit_t       *big_un;

1330                                small_un = (mm_unit32_od_t *)((uintptr_t)rbp +
1331                                    (sizeof (mddb_rb_t) - sizeof (int)));
1332                                newreqsize = sizeof (mm_unit_t);
1333                                big_un = (void *)Zalloc(newreqsize);
1334                                mirror_convert((caddr_t)small_un,
1335                                    (caddr_t)big_un, SMALL_2_BIG);
1336                                mdrec->record = (void *)big_un;
1337                        } else {
1338                                mm_unit_t       *big_un;

1340                                big_un = (mm_unit_t *)((uintptr_t)rbp +
1341                                    (sizeof (mddb_rb_t) - sizeof (int)));
1342                                newreqsize = sizeof (mm_unit_t);
1343                                mdrec->record = (void *)Zalloc(newreqsize);
1344                                bcopy(big_un, mdrec->record, newreqsize);
1345                        }
1346                        uc = ((mm_unit_t *)mdrec->record)->c;
1347                        mdrec->dfunc = &process_mirror;
1348                        mdrec->un_self_id = uc.un_self_id;
1349                        mdrec->has_parent = MD_HAS_PARENT(
1350                            uc.un_parent);
1351                        break;
1352                case MDDB_F_RAID:
1353                        if (is_32bit_record) {
1354                                mr_unit32_od_t  *small_un;
1355                                mr_unit_t       *big_un;
1356                                uint_t          ncol;

1358                                small_un = (mr_unit32_od_t *)((uintptr_t)rbp +
1359                                    (sizeof (mddb_rb_t) - sizeof (int)));
1360                                ncol = small_un->un_totalcolumncnt;
1361                                newreqsize = sizeof (mr_unit_t) +
1362                                    ((ncol - 1) * sizeof (mr_column_t));
1363                                big_un = (void *)Zalloc(newreqsize);
1364                                raid_convert((caddr_t)small_un,
1365                                    (caddr_t)big_un, SMALL_2_BIG);
1366                                mdrec->record = (void *)big_un;
1367                        } else {
1368                                mr_unit_t       *big_un;
1369                                uint_t          ncol;

1371                                big_un = (mr_unit_t *)((uintptr_t)rbp +
1372                                    (sizeof (mddb_rb_t) - sizeof (int)));
1373                                ncol = big_un->un_totalcolumncnt;
1374                                newreqsize = sizeof (mr_unit_t) +
1375                                    ((ncol - 1) * sizeof (mr_column_t));
1376                                mdrec->record = (void *)Zalloc(newreqsize);
1377                                bcopy(big_un, mdrec->record, newreqsize);
1378                        }
1379                        uc = ((mr_unit_t *)mdrec->record)->c;
1380                        mdrec->dfunc = &process_raid;
1381                        mdrec->un_self_id = uc.un_self_id;
1382                        mdrec->has_parent = MD_HAS_PARENT(
1383                            uc.un_parent);
1384                        break;
1385                case MDDB_F_TRANS_MASTER:
1386                        if (is_32bit_record) {
1387                                mt_unit32_od_t  *small_un;
```



```
1388                                mt_unit_t       *big_un;

1390                                small_un = (mt_unit32_od_t *)((uintptr_t)rbp +
1391                                    (sizeof (mddb_rb_t) - sizeof (int)));
1392                                newreqsize = sizeof (mt_unit_t);
1393                                big_un = (void *)Zalloc(newreqsize);
1394                                trans_master_convert((caddr_t)small_un,
1395                                    (caddr_t)big_un, SMALL_2_BIG);
1396                                mdrec->record = (void *)big_un;
1397                        } else {
1398                                mt_unit_t       *big_un;

1400                                big_un = (mt_unit_t *)((uintptr_t)rbp +
1401                                    (sizeof (mddb_rb_t) - sizeof (int)));
1402                                newreqsize = sizeof (mt_unit_t);
1403                                mdrec->record = (void *)Zalloc(newreqsize);
1404                                bcopy(big_un, mdrec->record, newreqsize);
1405                        }
1406                        uc = ((mt_unit_t *)mdrec->record)->c;
1407                        mdrec->dfunc = &process_trans;
1408                        mdrec->un_self_id = uc.un_self_id;
1409                        mdrec->has_parent = MD_HAS_PARENT(
1410                            uc.un_parent);
1411                        break;
1412                case MDDB_F_HOTSPARE:
1413                        if (is_32bit_record) {
1414                                hot_spare32_od_t        *small_un;
1415                                hot_spare_t             *big_un;

1417                                small_un = (hot_spare32_od_t *)((uintptr_t)rbp +
1418                                    (sizeof (mddb_rb_t) - sizeof (int)));
1419                                newreqsize = sizeof (hot_spare_t);
1420                                big_un = (void *)Zalloc(newreqsize);
1421                                hs_convert((caddr_t)small_un,
1422                                    (caddr_t)big_un, SMALL_2_BIG);
1423                                mdrec->record = (void *)big_un;
1424                        } else {
1425                                hot_spare_t             *big_un;

1427                                big_un = (hot_spare_t *)((uintptr_t)rbp +
1428                                    (sizeof (mddb_rb_t) - sizeof (int)));
1429                                newreqsize = sizeof (hot_spare_t);
1430                                mdrec->record = (void *)Zalloc(newreqsize);
1431                                bcopy(big_un, mdrec->record, newreqsize);
1432                        }
1433                        hs = (hot_spare_t *)mdrec->record;
1434                        mdrec->dfunc = &process_hotspare;
1435                        mdrec->un_self_id = NULL;
1436                        mdrec->hs_record_id = hs->hs_record_id;
1437                        mdrec->has_parent = 1;
1438                        break;
1439                case MDDB_F_HOTSPARE_POOL:
1440                        /*
1441                         * Ondisk and incore records are always same size.
1442                         */
1443                        hsp = (hot_spare_pool_ond_t *)((uintptr_t)rbp +
1444                            (sizeof (mddb_rb_t) - sizeof (int)));
1445                        newreqsize = sizeof (hot_spare_pool_ond_t) +
1446                            (sizeof (mddb_recid_t) * hsp->hsp_nhotspares);
1447                        mdrec->record = (void *)Zalloc(newreqsize);
1448                        bcopy(hsp, mdrec->record, newreqsize);
1449                        hsp = (hot_spare_pool_ond_t *)mdrec->record;
1450                        mdrec->dfunc = &process_hotspare_pool;
1451                        /*
1452                         * If the hsp has descriptive name we'll get
1453                         * the un_self_id
```

```
1454                         */
1455                        if (HSP_ID_IS_FN(hsp->hsp_self_id))
1456                                mdrec->un_self_id = hsp->hsp_self_id;
1457                        else
1458                                mdrec->un_self_id = NULL;
1459                        mdrec->has_parent = 0;
1460                        break;
1461                /* All valid cases have been dealt with */
1462                default:
1463                        (void) mdmddberror(ep, MDE_DB_NODB, 0, 0, 0, NULL);
1464                        return (-1);
1465        }

1467        /*
1468         * If metadevice record has an entry in the NM namespace
1469         * then it is copied into the mdrec->n_name field.
1470         */
1471        if (mdrec->un_self_id != NULL) {
1472                for (nmname = &nm_record->r_name[0]; nmname->n_key != 0;
1473                /*LINTED*/
1474                    nmname = (struct nm_name *)((char *)nmname +
1475                    NAMSIZ(nmname))) {
1476                        /*
1477                         * Extract the metadevice/hsp name if it is
1478                         * in the namespace.
1479                         *
1480                         * If it is a hot spare pool we will find our
1481                         * match by comparing the NM record's n_key
1482                         * with the extracted key from the hsp_self_id
1483                         * Else, match the un_self_id for the record
1484                         * to the n_minor name in the NM record.
1485                         */
1486                           if (mdrec->md_type == MDDB_F_HOTSPARE_POOL) {
1487                               if (nmname->n_key ==
1488                                   HSP_ID_TO_KEY(hsp->hsp_self_id)) {
1489                                       mdrec->n_key = nmname->n_key;
1490                                       uname = Strdup(nmname->n_name);
1491                                       mdrec->n_name = uname;
1492                                       break;
1493                               }
1494                           } else {
1495                               if ((nmname->n_minor) == (uc.un_self_id)) {
1496                                       (*mdrec).n_key = nmname->n_key;
1497                                       uname = Strdup(nmname->n_name);
1498                                       mdrec->n_name = uname;
1499                                       break;
1500                               }
1501                           }
1502                }
1503        }

1505        /*
1506         * If the metadevice name is not in the namespace, then
1507         * then we will generate the name from the minor number
1508         * for the metadevice.  In the case of records for a hotspare
1509         * pool we use hsp_self_id, otherwise we use un_self_id.
1510         */
1511        if (uname == NULL) {
1512                if (mdrec->md_type == MDDB_F_HOTSPARE_POOL) {
1513                        uname = Malloc(MAXSIZEMDRECNAME);
1514                        (void) sprintf(uname, "hsp%03u",
1515                            HSP_ID(hsp->hsp_self_id));
1516                        mdrec->n_name = uname;
1517                } else if (mdrec->md_type != MDDB_F_HOTSPARE) {
1518                        /*
1519                         * Generate the metadevice name for all other records
```



```
1520                         * (except for hotspares, because hotspares can only
1521                         * be physical devices.)
1522                         */
1523                        uname = Malloc(MAXSIZEMDRECNAME);
1524                        (void) sprintf(uname, "d%lu",
1525                            MD_MIN2UNIT(mdrec->un_self_id));
1526                        mdrec->n_name = uname;
1527                }
1528        }

1530        return (0);
1531 }


1534 /*
1535  * read_mdrecord()
1536  *
1537  * Reads the mddb_rb32_od_t or mddb_rb_t and the associated metadevice record
1538  * from the disk.  Runs magic, checksum, and revision checks on the record
1539  * block.
1540  *
1541  * Returns:
1542  *      < 0 for failure
1543  *        0 for success
1544  *
1545  */
1546 static int
1547 read_mdrecord(
1548         md_im_rec_t     **mdimpp,
1549         mddb_mb_t       *mbp,
1550         mddb_rb_t       *nm,
1551         mddb_de_t       *dep,
1552         char            *diskname,
1553         int             fd,
1554         md_timeval32_t  *lastaccess,
1555         md_error_t      *ep
1556 )
1557 {
1558         int             cnt, rval = 0;
1559         daddr_t         pblk;
1560         md_im_rec_t     *tmp_mdrec;
1561         void            *rbp = NULL;
1562         char            *rbp_tmp = NULL;
1563         mddb_rb32_t     *rbp_32;
1564         mddb_rb_t       *rbp_64;
1565         crc_skip_t      *skip = NULL;
1566         int             is_32bit_record;

1568         tmp_mdrec = Zalloc(sizeof (md_im_rec_t));
1569         rbp = (void *)Zalloc(dbtob(dep->de_blkcount));
1570         rbp_tmp = (char *)rbp;

1572         /* Read in the appropriate record and return configurations */
1573         for (cnt = 0; cnt < dep->de_blkcount; cnt++) {
1574                 if ((pblk = getphysblk(dep->de_blks[cnt], mbp)) < 0) {
1575                         rval = mdmddberror(ep, MDE_DB_BLKRANGE,
1576                             NODEV32, MD_LOCAL_SET,
1577                             dep->de_blks[cnt], diskname);
1578                         return (rval);
1579                 }

1581                 if (lseek(fd, (off_t)dbtob(pblk), SEEK_SET) < 0) {
1582                         rval = mdsyserror(ep, errno, diskname);
1583                         return (rval);
1584                 }
```

```
1586                if (read(fd, rbp_tmp, DEV_BSIZE) != DEV_BSIZE) {
1587                        rval = mdsyserror(ep, errno, diskname);
1588                        return (rval);
1589                }

1591                rbp_tmp += DEV_BSIZE;
1592        }
1593        tmp_mdrec->md_type = dep->de_flags;

1595        /*
1596         * The only place to discover whether or not the record is a
1597         * 32bit or 64bit record is from the record's rb_revision field.
1598         * The mddb_rb_t and mddb_rb32_t structures are identical for the
1599         * following fields:
1600         *      rb_magic, rb_revision, rb_checksum, and rb_checksum_fiddle.
1601         * So we can assume that the record is a 32bit structure when we
1602         * check the record's magic number and revision and when we calculate
1603         * the records checksum.
1604         */
1605        rbp_32 = (mddb_rb32_t *)rbp;

1607        /*
1608         * Checking the magic number for the record block.
1609         */
1610        if (rbp_32->rb_magic != MDDB_MAGIC_RB) {
1611                rval = -1;
1612                (void) mdmddberror(ep, MDE_DB_NODB, 0, 0, 0, NULL);
1613                goto out;
1614        }

1616        /*
1617         * Checking the revision for the record block. Must match either
1618         * revision for the current 64bit or 32bit record block.  Also,
1619         * setting the flag for whether or not it is a 32bit record.
1620         */
1621        is_32bit_record = 0;
1622        switch (rbp_32->rb_revision) {
1623        case MDDB_REV_RB:
1624        case MDDB_REV_RBFN:
1625                is_32bit_record = 1;
1626                break;
1627        case MDDB_REV_RB64:
1628        case MDDB_REV_RB64FN:
1629                break;
1630        default:
1631                rval = -1;
1632                (void) mdmddberror(ep, MDE_DB_NODB, 0, 0, 0, NULL);
1633                goto out;
1634        }

1636        /*
1637         * Calculating the checksum for this record block. Need
1638         * to skip the rb's checksum fiddle.
1639         */
1640        skip = (crc_skip_t *)Malloc(sizeof (crc_skip_t));
1641        skip->skip_next = NULL;
1642        skip->skip_offset = offsetof(mddb_rb_t, rb_checksum_fiddle);
1643        skip->skip_size = 3 * sizeof (uint_t);
1644        if (crcchk(rbp_32, &rbp_32->rb_checksum, dep->de_recsize, skip)) {
1645                rval = -1;
1646                (void) mdmddberror(ep, MDE_DB_NODB, 0, 0, 0, NULL);
1647                goto out;
1648        }

1650        /* mddb_rb_t and mddb_rb32_t differ before the rb_timestamp field */
1651        if (!is_32bit_record) {
```



```
1652                if ((*lastaccess).tv_sec < rbp_32->rb_timestamp.tv_sec) {
1653                    *lastaccess = rbp_32->rb_timestamp;
1654                } else if ((*lastaccess).tv_sec ==
1655                    rbp_32->rb_timestamp.tv_sec) {
1656                        if ((*lastaccess).tv_usec <
1657                            rbp_32->rb_timestamp.tv_usec)
1658                                *lastaccess = rbp_32->rb_timestamp;
1659                }
1660        } else {
1661                rbp_64 = (mddb_rb_t *)rbp;
1662                if ((*lastaccess).tv_sec < rbp_64->rb_timestamp.tv_sec) {
1663                    *lastaccess = rbp_64->rb_timestamp;
1664                } else if ((*lastaccess).tv_sec ==
1665                    rbp_64->rb_timestamp.tv_sec) {
1666                        if ((*lastaccess).tv_usec <
1667                            rbp_64->rb_timestamp.tv_usec)
1668                                *lastaccess = rbp_64->rb_timestamp;
1669                }
1670        }

1672        /* Populates the fields in md_im_rec_t *tmp_mdrec. */
1673        rval = extract_mduser_data(nm, tmp_mdrec, rbp, is_32bit_record, ep);
1674        if (rval < 0)
1675                goto out;

1677        /* Adding record to the head of the list of all metadevices. */
1678        tmp_mdrec->prev = NULL;
1679        if (*mdimpp == NULL) {
1680                tmp_mdrec->next = NULL;
1681                *mdimpp = tmp_mdrec;
1682        } else {
1683                (*mdimpp)->prev = tmp_mdrec;
1684                tmp_mdrec->next = *mdimpp;
1685                *mdimpp = tmp_mdrec;
1686        }

1688 out:
1689        /* Free the skip list */
1690        while (skip) {
1691                crc_skip_t      *skip_rm = skip;

1693                skip = skip->skip_next;
1694                Free(skip_rm);
1695        }

1697        if (rbp)
1698                Free(rbp);

1700        return (rval);
1701 }


1704 /*
1705  * read_all_mdrecords()
1706  *
1707  * Reads the directory block and directory entries.
1708  * Runs magic, checksum, and revision checks on the directory block.
1709  *
1710  * Returns:
1711  *      < 0 for failure
1712  *        0 for success
1713  */
1714 static int
1715 read_all_mdrecords(
1716         md_im_rec_t     **mdimpp,
1717         mddb_mb_t       *mbp,
```

```
        mddb_lb_t       *lbp,
        mddb_rb_t       *nm,
        mdname_t        *rsp,
        int             fd,
        md_timeval32_t  *lastaccess,
        md_error_t      *ep
)
{
        int             dbblk, rval = 0;
        char            db[DEV_BSIZE];
        mddb_de_t       *dep;
        int             desize;
        /*LINTED*/
        mddb_db_t       *dbp = (mddb_db_t *)&db;

        /* Read in all directory blocks */
        for (dbblk = lbp->lb_dbfirstblk;
            dbblk != 0;
            dbblk = dbp->db_nextblk) {

                if ((rval = read_database_block(ep, fd, mbp, dbblk,
                    dbp, sizeof (db))) <= 0)
                        goto out;

                /*
                 * Set ep with error code for MDE_DB_NODB.  This is the
                 * error code used in the kernel when there is a problem
                 * with reading records in.  Checks the magic number, the
                 * revision, and the checksum for each directory block.
                 */
                if (dbp->db_magic != MDDB_MAGIC_DB) {
                        rval = -1;
                        (void) mdmddberror(ep, MDE_DB_NODB, 0, 0, 0, NULL);
                        goto out;
                }

                if (revchk(MDDB_REV_DB, dbp->db_revision)) {
                        rval = -1;
                        (void) mdmddberror(ep, MDE_DB_NODB, 0, 0, 0, NULL);
                        goto out;
                }

                if (crcchk(dbp, &dbp->db_checksum, MDDB_BSIZE, NULL)) {
                        rval = -1;
                        (void) mdmddberror(ep, MDE_DB_NODB, 0, 0, 0, NULL);
                        goto out;
                }

                /*
                 * If db timestamp is more recent than the previously recorded
                 * last modified timestamp, then update last modified.
                 */
                if ((*lastaccess).tv_sec < dbp->db_timestamp.tv_sec) {
                        *lastaccess = dbp->db_timestamp;
                } else if ((*lastaccess).tv_sec == dbp->db_timestamp.tv_sec) {
                        if ((*lastaccess).tv_usec < dbp->db_timestamp.tv_usec)
                                *lastaccess = dbp->db_timestamp;
                }

                /* Creates dep list of all directory entries in the db */
                if (dbp->db_firstentry != NULL) {
                        /* LINTED */
                        dep = (mddb_de_t *)((caddr_t)(&dbp->db_firstentry)
                            + sizeof (dbp->db_firstentry));
                        dbp->db_firstentry = dep;
                        while (dep && dep->de_next) {
```



```
                                desize = sizeof (*dep) -
                                    sizeof (dep->de_blks) +
                                    sizeof (daddr_t) * dep->de_blkcount;
                                /* LINTED */
                                dep->de_next = (mddb_de_t *)
                                    ((caddr_t)dep + desize);
                                dep = dep->de_next;
                        }
                }

                /*
                 * Process all directory entries in the directory block.
                 * For each directory entry, read_mdrec is called to read
                 * in the record data.
                 */
                for (dep = dbp->db_firstentry; dep != NULL;
                    dep = dep->de_next) {

                        /*
                         * de_flags is set to the type of metadevice.
                         * If directory entry does not correspond to a
                         * specific metadevice then it is set to zero.
                         * All namespace records(NM, SHR_NM, DID_SHR_NM) have a
                         * value of zero in their de_flags field.
                         */
                        if ((dep->de_flags != 0)&&
                            (dep->de_flags != MDDB_F_OPT) &&
                            (dep->de_flags != MDDB_F_TRANS_LOG) &&
                            (dep->de_flags != MDDB_F_CHANGELOG)) {
                                rval = read_mdrecord(mdimpp, mbp, nm, dep,
                                    rsp->cname, fd, lastaccess, ep);
                                if (rval < 0)
                                        goto out;
                        }
                }
        }

out:
        return (rval);
}


/*
 * report_metastat_info()
 *
 * Generates the metastat -c output.  Also, updates the global variable
 * for a last accessed timestamp.
 *
 * Returns:
 *      < 0 for failure
 *        0 for success
 *
 */
int
report_metastat_info(
        mddb_mb_t               *mb,
        mddb_lb_t               *lbp,
        mddb_rb_t               *nm,
        pnm_rec_t               **pnm,
        mdname_t                *rsp,
        int                     fd,
        md_timeval32_t          *lastaccess,
        md_error_t              *ep
)
{
        int rval = 0;
```

```
1850         /* list of all metadevices in diskset */
1851         md_im_rec_t     *mdimp = NULL;
1852         md_im_rec_t     *tmp_mdrec, *rm_mdrec;

1854         /* Read in metadevice records and add entries to mdimp list. */
1855         rval = read_all_mdrecords(&mdimp, mb, lbp, nm, rsp, fd, lastaccess,
1856             ep);
1857         if (rval < 0)
1858                 goto out;

1860         /* Adding a fake record to the head of the list of all metadevices. */
1861         if (mdimp != NULL) {
1862                 tmp_mdrec = Zalloc(sizeof (md_im_rec_t));
1863                 tmp_mdrec->prev = NULL;
1864                 mdimp->prev = tmp_mdrec;
1865                 tmp_mdrec->next = mdimp;
1866                 mdimp = tmp_mdrec;
1867         }

1869         /* Calling functions to process all metadevices on mdimp list */
1870         process_toplevel_devices(&mdimp, *pnm, MDDB_F_SOFTPART);
1871         process_toplevel_devices(&mdimp, *pnm, MDDB_F_TRANS_MASTER);
1872         process_toplevel_devices(&mdimp, *pnm, MDDB_F_MIRROR);
1873         process_toplevel_devices(&mdimp, *pnm, MDDB_F_RAID);
1874         process_toplevel_devices(&mdimp, *pnm, MDDB_F_STRIPE);
1875         process_toplevel_devices(&mdimp, *pnm, MDDB_F_HOTSPARE_POOL);
1876         (void) printf("\n");

1878 out:
1879         /*
1880          * If mdreclist is not null, then this will walk through all
1881          * elements and free them.
1882          */
1883         tmp_mdrec = mdimp;
1884         while (tmp_mdrec != NULL) {
1885                 rm_mdrec = tmp_mdrec;
1886                 tmp_mdrec = tmp_mdrec->next;
1887                 if (rm_mdrec->record != NULL)
1888                         Free(rm_mdrec->record);
1889                 if (rm_mdrec->n_name != NULL)
1890                         Free(rm_mdrec->n_name);
1891                 Free(rm_mdrec);
1892         }

1894         free_pnm_rec_list(pnm);
1895         return (rval);
1896 }
```

****
1405 Thu Sep 5 23:02:10 2013
new/usr/src/stand/lib/sa/stddef.h
3373 gcc >= 4.5 concerns about offsetof()
****

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SA_STDDEF_H
#define _SA_STDDEF_H

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

/*
 * Exported interfaces for standalone's subset of libc's <stddef.h>.
 * All standalone code *must* use this header rather than libc's.
 */

#ifdef  __cplusplus
extern "C" {
#endif

#ifndef NULL
#define NULL    0
#endif

#ifndef offsetof
#if defined(__GNUC__)
#define offsetof(s, m)  __builtin_offsetof(s, m)
#else
#define offsetof(s, m)  ((size_t)(&(((s *)0)->m)))
#endif
#define offsetof(s, m)  (size_t)(&(((s *)0)->m))
#endif

#ifdef  __cplusplus
}
#endif
```

_____unchanged_portion_omitted_

```
**********************************************************
   26760 Thu Sep  5 23:02:11 2013
new/usr/src/uts/common/avs/ns/rdc/rdc.c
3373 gcc >= 4.5 concerns about offsetof()
**********************************************************
______unchanged_portion_omitted_


 525 /*
 526  * Yet another standard thing that is not standard ...
 527  */
 528 #ifndef offsetof
 529 #if defined(__GNUC__)
 530 #define offsetof(s, m)  __builtin_offsetof(s, m)
 531 #else
 532 #define offsetof(s, m)  ((size_t)(&(((s *)0)->m)))
 533 #endif
 529 #define offsetof(s, m)  ((size_t)(&((s *)0)->m))
 534 #endif

 536 /*
 537  * Build a 32bit rdc_set structure and copyout to the user level.
 538  */
 539 int
 540 rdc_status_copy32(const void *arg, void *usetp, size_t size, int mode)
 541 {
 542         rdc_u_info_t *urdc = (rdc_u_info_t *)arg;
 543         struct rdc_set32 set32;
 544         size_t tailsize;
 545 #ifdef DEBUG
 546         size_t tailsize32;
 547 #endif

 549         bzero(&set32, sizeof (set32));

 551         tailsize = sizeof (struct rdc_addr32) -
 552             offsetof(struct rdc_addr32, intf);

 554         /* primary address structure, avoiding netbuf */
 555         bcopy(&urdc->primary.intf[0], &set32.primary.intf[0], tailsize);

 557         /* secondary address structure, avoiding netbuf */
 558         bcopy(&urdc->secondary.intf[0], &set32.secondary.intf[0], tailsize);

 560         /*
 561          * the rest, avoiding netconfig
 562          * note: the tail must be the same size in both structures
 563          */
 564         tailsize = sizeof (struct rdc_set) - offsetof(struct rdc_set, flags);
 565 #ifdef DEBUG
 566         /*
 567          * ASSERT is calling for debug reason, and tailsize32 is only declared
 568          * for ASSERT, put them under debug to avoid lint warning.
 569          */
 570         tailsize32 = sizeof (struct rdc_set32) -
 571             offsetof(struct rdc_set32, flags);
 572         ASSERT(tailsize == tailsize32);
 573 #endif

 575         bcopy(&urdc->flags, &set32.flags, tailsize);

 577         /* copyout to user level */
 578         return (ddi_copyout(&set32, usetp, size, mode));
 579 }
_____unchanged_portion_omitted_
1119 #endif
```

```
**********************************************************
  161179 Thu Sep  5 23:02:12 2013
new/usr/src/uts/common/avs/ns/rdc/rdc_io.c
3373 gcc >= 4.5 concerns about offsetof()
**********************************************************
______unchanged_portion_omitted_

5642 /*
5643  * Yet another standard thing that is not standard ...
5644  */
5645 #ifndef offsetof
5646 #if defined(__GNUC__)
5647 #define offsetof(s, m)  __builtin_offsetof(s, m)
5648 #else
5649 #define offsetof(s, m)  ((size_t)(&(((s *)0)->m)))
5650 #endif
5646 #define offsetof(s, m)  ((size_t)(&((s *)0)->m))
5651 #endif

5653 static int
5654 rdc_status(void *arg, int mode, rdc_config_t *uparms, spcs_s_info_t kstatus)
5655 {
5656         rdc_k_info_t *krdc;
5657         rdc_u_info_t *urdc;
5658         disk_queue *dqp;
5659         int rc = 0;
5660         int index;
5661         char *ptr;
5662         extern int rdc_status_copy32(const void *, void *, size_t, int);

5664         mutex_enter(&rdc_conf_lock);
5665         index = rdc_lookup_byname(uparms->rdc_set);
5666         if (index >= 0)
5667                 krdc = &rdc_k_info[index];
5668         if (index < 0 || (krdc->type_flag & RDC_DISABLEPEND)) {
5669                 mutex_exit(&rdc_conf_lock);
5670                 spcs_s_add(kstatus, RDC_EALREADY, uparms->rdc_set->primary.file,
5671                     uparms->rdc_set->secondary.file);
5672                 return (RDC_EALREADY);
5673         }

5675         set_busy(krdc);
5676         if (krdc->type_flag == 0) {
5677                 /* A resume or enable failed */
5678                 wakeup_busy(krdc);
5679                 mutex_exit(&rdc_conf_lock);
5680                 spcs_s_add(kstatus, RDC_EALREADY, uparms->rdc_set->primary.file,
5681                     uparms->rdc_set->secondary.file);
5682                 return (RDC_EALREADY);
5683         }

5685         mutex_exit(&rdc_conf_lock);

5687         rdc_group_enter(krdc);
5688         if (rdc_check(krdc, uparms->rdc_set)) {
5689                 rdc_group_exit(krdc);
5690                 spcs_s_add(kstatus, RDC_EALREADY, uparms->rdc_set->primary.file,
5691                     uparms->rdc_set->secondary.file);
5692                 rc = RDC_EALREADY;
5693                 goto done;
5694         }

5696         urdc = &rdc_u_info[index];

5698         /*
5699          * sneak out qstate in urdc->flags
```



```
5700          * this is harmless because it's value is not used
5701          * in urdc->flags. the real qstate is kept in
5702          * group->diskq->disk_hdr.h.state
5703          */
5704         if (RDC_IS_DISKQ(krdc->group)) {
5705                 dqp = &krdc->group->diskq;
5706                 if (IS_QSTATE(dqp, RDC_QNOBLOCK))
5707                 urdc->flags |= RDC_QNOBLOCK;
5708         }

5710         if (ddi_model_convert_from(mode & FMODELS) == DDI_MODEL_ILP32) {
5711                 ptr = (char *)arg + offsetof(struct rdc_config32, rdc_set);
5712                 rc = rdc_status_copy32(urdc, ptr, sizeof (struct rdc_set32),
5713                     mode);
5714         } else {
5715                 ptr = (char *)arg + offsetof(struct rdc_config, rdc_set);
5716                 rc = ddi_copyout(urdc, ptr, sizeof (struct rdc_set), mode);
5717         }
5718         /* clear out qstate from flags */
5719         urdc->flags &= ~RDC_QNOBLOCK;

5721         if (rc)
5722                 rc = EFAULT;

5724         rdc_group_exit(krdc);
5725 done:
5726         mutex_enter(&rdc_conf_lock);
5727         wakeup_busy(krdc);
5728         mutex_exit(&rdc_conf_lock);

5730         return (rc);
5731 }
______unchanged_portion_omitted_
```

```
**********************************************************
   60867 Thu Sep  5 23:02:13 2013
new/usr/src/uts/common/avs/ns/sv/sv.c
3373 gcc >= 4.5 concerns about offsetof()
**********************************************************
______unchanged_portion_omitted_


#ifndef offsetof
#if defined(__GNUC__)
#define offsetof(s, m)  __builtin_offsetof(s, m)
#else
#define offsetof(s, m)  ((size_t)(&(((s *)0)->m)))
#endif
#define offsetof(s, m)  ((size_t)(&((s *)0)->m))
#endif

/*
 * re-write the size of the current partition
 */
static int
sv_fix_dkiocgvtoc(const intptr_t arg, const int mode, sv_dev_t *svp)
{
        size_t offset;
        int ilp32;
        int pnum;
        int rc;

        ilp32 = (ddi_model_convert_from((mode & FMODELS)) == DDI_MODEL_ILP32);

        rc = nskern_partition(svp->sv_dev, &pnum);
        if (rc != 0) {
                return (rc);
        }

        if (pnum < 0 || pnum >= V_NUMPAR) {
                cmn_err(CE_WARN,
                    "!sv_gvtoc: unable to determine partition number "
                    "for dev %lx", svp->sv_dev);
                return (EINVAL);
        }

        if (ilp32) {
                int32_t p_size;

#ifdef _SunOS_5_6
                offset = offsetof(struct vtoc, v_part);
                offset += sizeof (struct partition) * pnum;
                offset += offsetof(struct partition, p_size);
#else
                offset = offsetof(struct vtoc32, v_part);
                offset += sizeof (struct partition32) * pnum;
                offset += offsetof(struct partition32, p_size);
#endif

                p_size = (int32_t)svp->sv_nblocks;
                if (p_size == 0) {
                        if (sv_reserve(svp->sv_fd,
                            NSC_MULTI|NSC_PCATCH) == 0) {
                                p_size = (int32_t)svp->sv_nblocks;
                                nsc_release(svp->sv_fd);
                        } else {
                                rc = EINTR;
                        }
                }
```



```
                if ((rc == 0) && ddi_copyout(&p_size, (void *)(arg + offset),
                    sizeof (p_size), mode) != 0) {
                        rc = EFAULT;
                }
        } else {
                long p_size;

                offset = offsetof(struct vtoc, v_part);
                offset += sizeof (struct partition) * pnum;
                offset += offsetof(struct partition, p_size);

                p_size = (long)svp->sv_nblocks;
                if (p_size == 0) {
                        if (sv_reserve(svp->sv_fd,
                            NSC_MULTI|NSC_PCATCH) == 0) {
                                p_size = (long)svp->sv_nblocks;
                                nsc_release(svp->sv_fd);
                        } else {
                                rc = EINTR;
                        }
                }

                if ((rc == 0) && ddi_copyout(&p_size, (void *)(arg + offset),
                    sizeof (p_size), mode) != 0) {
                        rc = EFAULT;
                }
        }

        return (rc);
}
______unchanged_portion_omitted_
```

```
**********************************************************
   33850 Thu Sep  5 23:02:14 2013
new/usr/src/uts/common/io/vscan/vscan_svc.c
3373 gcc >= 4.5 concerns about offsetof()
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/stat.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/time.h>
#include <sys/varargs.h>
#include <sys/conf.h>
#include <sys/modctl.h>
#include <sys/cmn_err.h>
#include <sys/vnode.h>
#include <fs/fs_subr.h>
#include <sys/types.h>
#include <sys/file.h>
#include <sys/disp.h>
#include <sys/sdt.h>
#include <sys/cred.h>
#include <sys/list.h>
#include <sys/vscan.h>

#define VS_REQ_MAGIC            0x52515354 /* 'RQST' */

#define VS_REQS_DEFAULT         20000   /* pending scan requests - reql */
#define VS_NODES_DEFAULT        128     /* concurrent file scans */
#define VS_WORKERS_DEFAULT      32      /* worker threads */
#define VS_SCANWAIT_DEFAULT     15*60   /* seconds to wait for scan result */
#define VS_REQL_HANDLER_TIMEOUT 30
#define VS_EXT_RECURSE_DEPTH    8

/* access derived from scan result (VS_STATUS_XXX) and file attributes */
#define VS_ACCESS_UNDEFINED     0
#define VS_ACCESS_ALLOW         1       /* return 0 */
#define VS_ACCESS_DENY          2       /* return EACCES */

#define tolower(C)      (((C) >= 'A' && (C) <= 'Z') ? (C) - 'A' + 'a' : (C))
#if defined(__GNUC__)
#define offsetof(s, m)  __builtin_offsetof(s, m)
```



```
#else
#define offsetof(s, m)  ((size_t)(&(((s *)0)->m)))
#endif
#define offsetof(s, m)  (size_t)(&(((s *)0)->m))

/* global variables - tunable via /etc/system */
uint32_t vs_reqs_max = VS_REQS_DEFAULT; /* max scan requests */
uint32_t vs_nodes_max = VS_NODES_DEFAULT; /* max in-progress scan requests */
uint32_t vs_workers = VS_WORKERS_DEFAULT; /* max workers send reqs to vscand */
uint32_t vs_scan_wait = VS_SCANWAIT_DEFAULT; /* secs to wait for scan result */


/*
 * vscan_svc_state
 *
 *   +-----------------+
 *   | VS_SVC_UNCONFIG |
 *   +-----------------+
 *      |           ^
 *      | svc_init  | svc_fini
 *      v           |
 *   +-----------------+
 *   | VS_SVC_IDLE     |<----|
 *   +-----------------+     |
 *      |                    |
 *      | svc_enable         |
 *      |<-----------------|  |
 *      v                  |  |
 *   +-----------------+   |  |
 *   | VS_SVC_ENABLED  |--|  |
 *   +-----------------+      |
 *      |                     |
 *      | svc_disable         | handler thread exit,
 *      v                     | all requests complete
 *   +-----------------+      |
 *   | VS_SVC_DISABLED |-----|
 *   +-----------------+
 *
 * svc_enable may occur when we are already in the ENABLED
 * state if vscand has exited without clean shutdown and
 * then reconnected within the delayed disable time period
 * (vs_reconnect_timeout) - see vscan_drv
 */

typedef enum {
        VS_SVC_UNCONFIG,
        VS_SVC_IDLE,
        VS_SVC_ENABLED, /* service enabled and registered */
        VS_SVC_DISABLED /* service disabled and nunregistered */
} vscan_svc_state_t;
______unchanged_portion_omitted_
```

```
**********************************************************
   17218 Thu Sep  5 23:02:14 2013
new/usr/src/uts/common/sys/ecppvar.h
3373 gcc >= 4.5 concerns about offsetof()
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_ECPPVAR_H
#define _SYS_ECPPVAR_H

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

#include <sys/note.h>

#ifdef  __cplusplus
extern "C" {
#endif

struct ecppunit;

/*
 * Hardware-abstraction structure
 */
struct ecpp_hw {
        int     (*map_regs)(struct ecppunit *);         /* map registers */
        void    (*unmap_regs)(struct ecppunit *);       /* unmap registers */
        int     (*config_chip)(struct ecppunit *);      /* configure SuperIO */
        void    (*config_mode)(struct ecppunit *);      /* config new mode */
        void    (*mask_intr)(struct ecppunit *);        /* mask interrupts */
        void    (*unmask_intr)(struct ecppunit *);      /* unmask interrupts */
        int     (*dma_start)(struct ecppunit *);        /* start DMA transfer */
        int     (*dma_stop)(struct ecppunit *, size_t *); /* stop DMA xfer */
        size_t  (*dma_getcnt)(struct ecppunit *);       /* get DMA counter */
        ddi_dma_attr_t  *attr;                          /* DMA attributes */
};
```

*unchanged_portion_omitted_*

```
/*
 * Macros for Cheerio/RIO DMAC programming
 */
#define SET_DMAC_CSR(pp, val)   ddi_put32(pp->uh.ebus.d_handle, \
                                ((uint32_t *)&pp->uh.ebus.dmac->csr), \
                                ((uint32_t)val))
#define GET_DMAC_CSR(pp)        ddi_get32(pp->uh.ebus.d_handle, \
                                (uint32_t *)&(pp->uh.ebus.dmac->csr))

#define SET_DMAC_ACR(pp, val)   ddi_put32(pp->uh.ebus.d_handle, \
                                ((uint32_t *)&pp->uh.ebus.dmac->acr), \
                                ((uint32_t)val))

#define GET_DMAC_ACR(pp)        ddi_get32(pp->uh.ebus.d_handle, \
                                (uint32_t *)&pp->uh.ebus.dmac->acr)

#define SET_DMAC_BCR(pp, val)   ddi_put32(pp->uh.ebus.d_handle, \
                                ((uint32_t *)&pp->uh.ebus.dmac->bcr), \
                                ((uint32_t)val))

#define GET_DMAC_BCR(pp)        ddi_get32(pp->uh.ebus.d_handle, \
                                ((uint32_t *)&pp->uh.ebus.dmac->bcr))

#define DMAC_RESET_TIMEOUT      10000   /* in usec */

/*
 * Macros to distinguish between PIO and DMA Compatibility mode
 */
#define COMPAT_PIO(pp) (((pp)->io_mode == ECPP_PIO) &&          \
                    ((pp)->current_mode == ECPP_CENTRONICS ||   \
                    (pp)->current_mode == ECPP_COMPAT_MODE))

#define COMPAT_DMA(pp) (((pp)->io_mode == ECPP_DMA) &&          \
                    ((pp)->current_mode == ECPP_CENTRONICS ||   \
                    (pp)->current_mode == ECPP_COMPAT_MODE))

/*
 * Other useful macros
 */
#define NELEM(a)        (sizeof (a) / sizeof (*(a)))
#if defined(__GNUC__)
#define offsetof(s, m)  __builtin_offsetof(s, m)
#else
#endif /* ! codereview */
#define offsetof(s, m)  ((size_t)(&(((s *)0)->m)))
#endif
#endif /* ! codereview */

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_ECPPVAR_H */
```

```
**********************************************************
   10227 Thu Sep  5 23:02:15 2013
new/usr/src/uts/common/sys/ib/clients/of/sol_ofs/sol_cma.h
3373 gcc >= 4.5 concerns about offsetof()
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
 */

#ifndef _SYS_IB_CLIENTS_OF_SOL_OFS_SOL_CMA_H
#define _SYS_IB_CLIENTS_OF_SOL_OFS_SOL_CMA_H

#ifdef __cplusplus
extern "C" {
#endif


#include <sys/ib/clients/of/sol_ofs/sol_ofs_common.h>
#include <sys/ib/clients/of/rdma/rdma_cm.h>
#include <sys/ib/clients/of/sol_ofs/sol_ib_cma.h> /* Transport Specific */

#if !defined(offsetof)
#if defined(__GNUC__)
#define offsetof(s, m)  __builtin_offsetof(s, m)
#else
#define offsetof(s, m)  ((size_t)(&(((s *)0)->m)))
#endif
#define offsetof(s, m)          (size_t)(&(((s *)0)->m))
#endif

#define IS_UDP_CMID(idp)        ((idp)->ps == RDMA_PS_UDP || \
        (idp)->ps == RDMA_PS_IPOIB)
#define IS_VALID_SOCKADDR(sockaddrp) \
        ((sockaddrp)->sa_family == AF_INET || \
        (sockaddrp)->sa_family == AF_INET6)

/*
 * Global structure which contains information about all
 * CMIDs, which have called rdma_listen().
 */
typedef struct sol_cma_glbl_listen_s {
        avl_node_t      cma_listen_node;

        uint64_t        cma_listen_chan_sid;
        void            *cma_listen_clnt_hdl;
```



```
        void            *cma_listen_svc_hdl;
        genlist_t       cma_listen_chan_list;
} sol_cma_glbl_listen_t;
______unchanged_portion_omitted_
```

```
**********************************************************
   12340 Thu Sep  5 23:02:16 2013
new/usr/src/uts/common/sys/sysmacros.h
3373 gcc >= 4.5 concerns about offsetof()
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
/*        All Rights Reserved   */


/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright 2011, 2012 Nexenta Systems, Inc.  All rights reserved.
 */

#ifndef _SYS_SYSMACROS_H
#define _SYS_SYSMACROS_H

#include <sys/param.h>

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * Some macros for units conversion
 */
/*
 * Disk blocks (sectors) and bytes.
 */
#define dtob(DD)        ((DD) << DEV_BSHIFT)
#define btod(BB)        (((BB) + DEV_BSIZE - 1) >> DEV_BSHIFT)
#define btodt(BB)       ((BB) >> DEV_BSHIFT)
#define lbtod(BB)       (((offset_t)(BB) + DEV_BSIZE - 1) >> DEV_BSHIFT)

/* common macros */
#ifndef MIN
#define MIN(a, b)       ((a) < (b) ? (a) : (b))
#endif
#ifndef MAX
#define MAX(a, b)       ((a) < (b) ? (b) : (a))
#endif
#ifndef ABS
#define ABS(a)          ((a) < 0 ? -(a) : (a))
#endif
```



```
#ifndef SIGNOF
#define SIGNOF(a)       ((a) < 0 ? -1 : (a) > 0)
#endif

#ifdef _KERNEL

/*
 * Convert a single byte to/from binary-coded decimal (BCD).
 */
extern unsigned char byte_to_bcd[256];
extern unsigned char bcd_to_byte[256];

#define BYTE_TO_BCD(x)  byte_to_bcd[(x) & 0xff]
#define BCD_TO_BYTE(x)  bcd_to_byte[(x) & 0xff]

#endif  /* _KERNEL */

/*
 * WARNING: The device number macros defined here should not be used by device
 * drivers or user software. Device drivers should use the device functions
 * defined in the DDI/DKI interface (see also ddi.h). Application software
 * should make use of the library routines available in makedev(3). A set of
 * new device macros are provided to operate on the expanded device number
 * format supported in SVR4. Macro versions of the DDI device functions are
 * provided for use by kernel proper routines only. Macro routines bmajor(),
 * major(), minor(), emajor(), eminor(), and makedev() will be removed or
 * their definitions changed at the next major release following SVR4.
 */

#define O_BITSMAJOR     7       /* # of SVR3 major device bits */
#define O_BITSMINOR     8       /* # of SVR3 minor device bits */
#define O_MAXMAJ        0x7f    /* SVR3 max major value */
#define O_MAXMIN        0xff    /* SVR3 max minor value */


#define L_BITSMAJOR32   14      /* # of SVR4 major device bits */
#define L_BITSMINOR32   18      /* # of SVR4 minor device bits */
#define L_MAXMAJ32      0x3fff  /* SVR4 max major value */
#define L_MAXMIN32      0x3ffff /* MAX minor for 3b2 software drivers. */
                                /* For 3b2 hardware devices the minor is */
                                /* restricted to 256 (0-255) */

#ifdef _LP64
#define L_BITSMAJOR     32      /* # of major device bits in 64-bit Solaris */
#define L_BITSMINOR     32      /* # of minor device bits in 64-bit Solaris */
#define L_MAXMAJ        0xffffffffUL    /* max major value */
#define L_MAXMIN        0xffffffffUL    /* max minor value */
#else
#define L_BITSMAJOR     L_BITSMAJOR32
#define L_BITSMINOR     L_BITSMINOR32
#define L_MAXMAJ        L_MAXMAJ32
#define L_MAXMIN        L_MAXMIN32
#endif

#ifdef _KERNEL

/* major part of a device internal to the kernel */

#define major(x)        (major_t)((((unsigned)(x)) >> O_BITSMINOR) & O_MAXMAJ)
#define bmajor(x)       (major_t)((((unsigned)(x)) >> O_BITSMINOR) & O_MAXMAJ)

/* get internal major part of expanded device number */

#define getmajor(x)     (major_t)((((dev_t)(x)) >> L_BITSMINOR) & L_MAXMAJ)

/* minor part of a device internal to the kernel */
```

```
129 #define minor(x)        (minor_t)((x) & O_MAXMIN)

131 /* get internal minor part of expanded device number */

133 #define getminor(x)     (minor_t)((x) & L_MAXMIN)

135 #else

137 /* major part of a device external from the kernel (same as emajor below) */

139 #define major(x)        (major_t)((((unsigned)(x)) >> O_BITSMINOR) & O_MAXMAJ)

141 /* minor part of a device external from the kernel  (same as eminor below) */

143 #define minor(x)        (minor_t)((x) & O_MAXMIN)

145 #endif  /* _KERNEL */

147 /* create old device number */

149 #define makedev(x, y) (unsigned short)(((x) << O_BITSMINOR) | ((y) & O_MAXMIN))

151 /* make an new device number */

153 #define makedevice(x, y) (dev_t)(((dev_t)(x) << L_BITSMINOR) | ((y) & L_MAXMIN))


156 /*
157  * emajor() allows kernel/driver code to print external major numbers
158  * eminor() allows kernel/driver code to print external minor numbers
159  */

161 #define emajor(x) \
162         (major_t)(((unsigned int)(x) >> O_BITSMINOR) > O_MAXMAJ) ? \
163             NODEV : (((unsigned int)(x) >> O_BITSMINOR) & O_MAXMAJ)

165 #define eminor(x) \
166         (minor_t)((x) & O_MAXMIN)

168 /*
169  * get external major and minor device
170  * components from expanded device number
171  */
172 #define getemajor(x)    (major_t)((((dev_t)(x) >> L_BITSMINOR) > L_MAXMAJ) ? \
173                             NODEV : (((dev_t)(x) >> L_BITSMINOR) & L_MAXMAJ))
174 #define geteminor(x)    (minor_t)((x) & L_MAXMIN)

176 /*
177  * These are versions of the kernel routines for compressing and
178  * expanding long device numbers that don't return errors.
179  */
180 #if (L_BITSMAJOR32 == L_BITSMAJOR) && (L_BITSMINOR32 == L_BITSMINOR)

182 #define DEVCMPL(x)      (x)
183 #define DEVEXPL(x)      (x)

185 #else

187 #define DEVCMPL(x)      \
188         (dev32_t)((((x) >> L_BITSMINOR) > L_MAXMAJ32 || \
189             ((x) & L_MAXMIN) > L_MAXMIN32) ? NODEV32 : \
190             ((((x) >> L_BITSMINOR) << L_BITSMINOR32) | ((x) & L_MAXMIN32)))

192 #define DEVEXPL(x)      \
193         (((x) == NODEV32) ? NODEV : \
```



```
194         makedevice(((x) >> L_BITSMINOR32) & L_MAXMAJ32, (x) & L_MAXMIN32))

196 #endif /* L_BITSMAJOR32 ... */

198 /* convert to old (SVR3.2) dev format */

200 #define cmpdev(x) \
201         (o_dev_t)((((x) >> L_BITSMINOR) > O_MAXMAJ || \
202             ((x) & L_MAXMIN) > O_MAXMIN) ? NODEV : \
203             ((((x) >> L_BITSMINOR) << O_BITSMINOR) | ((x) & O_MAXMIN)))

205 /* convert to new (SVR4) dev format */

207 #define expdev(x) \
208         (dev_t)(((dev_t)((x) >> O_BITSMINOR) & O_MAXMAJ) << L_BITSMINOR) | \
209             ((x) & O_MAXMIN))

211 /*
212  * Macro for checking power of 2 address alignment.
213  */
214 #define IS_P2ALIGNED(v, a) ((((uintptr_t)(v)) & ((uintptr_t)(a) - 1)) == 0)

216 /*
217  * Macros for counting and rounding.
218  */
219 #define howmany(x, y)   (((x)+((y)-1))/(y))
220 #define roundup(x, y)   ((((x)+((y)-1))/(y))*(y))

222 /*
223  * Macro to determine if value is a power of 2
224  */
225 #define ISP2(x)         (((x) & ((x) - 1)) == 0)

227 /*
228  * Macros for various sorts of alignment and rounding.  The "align" must
229  * be a power of 2.  Often times it is a block, sector, or page.
230  */

232 /*
233  * return x rounded down to an align boundary
234  * eg, P2ALIGN(1200, 1024) == 1024 (1*align)
235  * eg, P2ALIGN(1024, 1024) == 1024 (1*align)
236  * eg, P2ALIGN(0x1234, 0x100) == 0x1200 (0x12*align)
237  * eg, P2ALIGN(0x5600, 0x100) == 0x5600 (0x56*align)
238  */
239 #define P2ALIGN(x, align)               ((x) & -(align))

241 /*
242  * return x % (mod) align
243  * eg, P2PHASE(0x1234, 0x100) == 0x34 (x-0x12*align)
244  * eg, P2PHASE(0x5600, 0x100) == 0x00 (x-0x56*align)
245  */
246 #define P2PHASE(x, align)               ((x) & ((align) - 1))

248 /*
249  * return how much space is left in this block (but if it's perfectly
250  * aligned, return 0).
251  * eg, P2NPHASE(0x1234, 0x100) == 0xcc (0x13*align-x)
252  * eg, P2NPHASE(0x5600, 0x100) == 0x00 (0x56*align-x)
253  */
254 #define P2NPHASE(x, align)              (-(x) & ((align) - 1))

256 /*
257  * return x rounded up to an align boundary
258  * eg, P2ROUNDUP(0x1234, 0x100) == 0x1300 (0x13*align)
259  * eg, P2ROUNDUP(0x5600, 0x100) == 0x5600 (0x56*align)
```

```
 */
#define P2ROUNDUP(x, align)             (-(-(x) & -(align)))

/*
 * return the ending address of the block that x is in
 * eg, P2END(0x1234, 0x100) == 0x12ff (0x13*align - 1)
 * eg, P2END(0x5600, 0x100) == 0x56ff (0x57*align - 1)
 */
#define P2END(x, align)                 (-(~(x) & -(align)))

/*
 * return x rounded up to the next phase (offset) within align.
 * phase should be < align.
 * eg, P2PHASEUP(0x1234, 0x100, 0x10) == 0x1310 (0x13*align + phase)
 * eg, P2PHASEUP(0x5600, 0x100, 0x10) == 0x5610 (0x56*align + phase)
 */
#define P2PHASEUP(x, align, phase)      ((phase) - (((phase) - (x)) & -(align)))

/*
 * return TRUE if adding len to off would cause it to cross an align
 * boundary.
 * eg, P2BOUNDARY(0x1234, 0xe0, 0x100) == TRUE (0x1234 + 0xe0 == 0x1314)
 * eg, P2BOUNDARY(0x1234, 0x50, 0x100) == FALSE (0x1234 + 0x50 == 0x1284)
 */
#define P2BOUNDARY(off, len, align) \
        (((off) ^ ((off) + (len) - 1)) > (align) - 1)

/*
 * Return TRUE if they have the same highest bit set.
 * eg, P2SAMEHIGHBIT(0x1234, 0x1001) == TRUE (the high bit is 0x1000)
 * eg, P2SAMEHIGHBIT(0x1234, 0x3010) == FALSE (high bit of 0x3010 is 0x2000)
 */
#define P2SAMEHIGHBIT(x, y)             (((x) ^ (y)) < ((x) & (y)))

/*
 * Typed version of the P2* macros.  These macros should be used to ensure
 * that the result is correctly calculated based on the data type of (x),
 * which is passed in as the last argument, regardless of the data
 * type of the alignment.  For example, if (x) is of type uint64_t,
 * and we want to round it up to a page boundary using "PAGESIZE" as
 * the alignment, we can do either
 *      P2ROUNDUP(x, (uint64_t)PAGESIZE)
 * or
 *      P2ROUNDUP_TYPED(x, PAGESIZE, uint64_t)
 */
#define P2ALIGN_TYPED(x, align, type)   \
        ((type)(x) & -(type)(align))
#define P2PHASE_TYPED(x, align, type)   \
        ((type)(x) & ((type)(align) - 1))
#define P2NPHASE_TYPED(x, align, type)  \
        (-(type)(x) & ((type)(align) - 1))
#define P2ROUNDUP_TYPED(x, align, type) \
        (-(-(type)(x) & -(type)(align)))
#define P2END_TYPED(x, align, type)     \
        (-(~(type)(x) & -(type)(align)))
#define P2PHASEUP_TYPED(x, align, phase, type)  \
        ((type)(phase) - (((type)(phase) - (type)(x)) & -(type)(align)))
#define P2CROSS_TYPED(x, y, align, type)        \
        (((type)(x) ^ (type)(y)) > (type)(align) - 1)
#define P2SAMEHIGHBIT_TYPED(x, y, type) \
        (((type)(x) ^ (type)(y)) < ((type)(x) & (type)(y)))

/*
 * Macros to atomically increment/decrement a variable.  mutex and var
 * must be pointers.
 */
```



```
#define INCR_COUNT(var, mutex) mutex_enter(mutex), (*(var))++, mutex_exit(mutex)
#define DECR_COUNT(var, mutex) mutex_enter(mutex), (*(var))--, mutex_exit(mutex)

/*
 * Macros to declare bitfields - the order in the parameter list is
 * Low to High - that is, declare bit 0 first.  We only support 8-bit bitfields
 * because if a field crosses a byte boundary it's not likely to be meaningful
 * without reassembly in its nonnative endianness.
 */
#if defined(_BIT_FIELDS_LTOH)
#define DECL_BITFIELD2(_a, _b)                          \
        uint8_t _a, _b
#define DECL_BITFIELD3(_a, _b, _c)                      \
        uint8_t _a, _b, _c
#define DECL_BITFIELD4(_a, _b, _c, _d)                  \
        uint8_t _a, _b, _c, _d
#define DECL_BITFIELD5(_a, _b, _c, _d, _e)              \
        uint8_t _a, _b, _c, _d, _e
#define DECL_BITFIELD6(_a, _b, _c, _d, _e, _f)          \
        uint8_t _a, _b, _c, _d, _e, _f
#define DECL_BITFIELD7(_a, _b, _c, _d, _e, _f, _g)      \
        uint8_t _a, _b, _c, _d, _e, _f, _g
#define DECL_BITFIELD8(_a, _b, _c, _d, _e, _f, _g, _h)  \
        uint8_t _a, _b, _c, _d, _e, _f, _g, _h
#elif defined(_BIT_FIELDS_HTOL)
#define DECL_BITFIELD2(_a, _b)                          \
        uint8_t _b, _a
#define DECL_BITFIELD3(_a, _b, _c)                      \
        uint8_t _c, _b, _a
#define DECL_BITFIELD4(_a, _b, _c, _d)                  \
        uint8_t _d, _c, _b, _a
#define DECL_BITFIELD5(_a, _b, _c, _d, _e)              \
        uint8_t _e, _d, _c, _b, _a
#define DECL_BITFIELD6(_a, _b, _c, _d, _e, _f)          \
        uint8_t _f, _e, _d, _c, _b, _a
#define DECL_BITFIELD7(_a, _b, _c, _d, _e, _f, _g)      \
        uint8_t _g, _f, _e, _d, _c, _b, _a
#define DECL_BITFIELD8(_a, _b, _c, _d, _e, _f, _g, _h)  \
        uint8_t _h, _g, _f, _e, _d, _c, _b, _a
#else
#error  One of _BIT_FIELDS_LTOH or _BIT_FIELDS_HTOL must be defined
#endif  /* _BIT_FIELDS_LTOH */

/* avoid any possibility of clashing with <stddef.h> version */
#if defined(_KERNEL) && !defined(_KMEMUSER)

#if !defined(offsetof)
#if defined(__GNUC__)
#define offsetof(s, m)  __builtin_offsetof(s, m)
#else
#endif /* ! codereview */
#define offsetof(s, m)  ((size_t)(&(((s *)0)->m)))
#endif
#endif /* ! codereview */
#endif /* !offsetof */

#define container_of(m, s, name)                        \
        (void *)((uintptr_t)(m) - (uintptr_t)offsetof(s, name))

#define ARRAY_SIZE(x)   (sizeof (x) / sizeof (x[0]))
#endif /* _KERNEL, !_KMEMUSER */

#ifdef  __cplusplus
}
#endif
```

```
392 #endif  /* _SYS_SYSMACROS_H */
```

```
**********************************************************
   10329 Thu Sep  5 23:02:17 2013
new/usr/src/uts/common/sys/usb/clients/audio/usb_ac/usb_ac.h
3373 gcc >= 4.5 concerns about offsetof()
**********************************************************
______unchanged_portion_omitted_

 266 /* warlock directives, stable data */
 267 _NOTE(MUTEX_PROTECTS_DATA(usb_ac_state_t::usb_ac_mutex, usb_ac_state_t))
 268 _NOTE(MUTEX_PROTECTS_DATA(usb_ac_state_t::usb_ac_mutex, usb_ac_power_t))
 269 _NOTE(MUTEX_PROTECTS_DATA(usb_ac_state_t::usb_ac_mutex, usb_ac_plumbed_t))
 270 _NOTE(MUTEX_PROTECTS_DATA(usb_audio_eng_t::lock, usb_audio_eng_t))
 271 _NOTE(MUTEX_PROTECTS_DATA(usb_audio_eng_t::lock, usb_audio_format_t))
 272 _NOTE(MUTEX_PROTECTS_DATA(usb_audio_ctrl_t::ctrl_mutex, usb_audio_ctrl_t))


 275 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_ac_state_t::usb_ac_dip))
 276 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_ac_state_t::usb_ac_ser_acc))
 277 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_ac_state_t::usb_ac_pm))
 278 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_ac_state_t::usb_ac_instance))
 279 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_ac_state_t::usb_ac_default_ph))
 280 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_ac_state_t::usb_ac_log_handle))
 281 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_ac_state_t::usb_ac_if_descr))
 282 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_ac_state_t::usb_ac_dev_data))
 283 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_ac_state_t::usb_ac_ifno))
 284 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_ac_state_t::flags))
 285 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_ac_state_t::usb_ac_input_ports))
 286 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_ac_state_t::engines))
 287 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_ac_state_t::usb_ac_audio_dev))
 288 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_ac_state_t::controls))

 290 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_audio_eng_t::af_eflags))
 291 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_audio_eng_t::streams))
 292 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_audio_eng_t::statep))
 293 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_audio_eng_t::fmt))
 294 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_audio_eng_t::fragfr))
 295 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_audio_eng_t::frsmshift))
 296 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_audio_eng_t::started))
 297 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_audio_eng_t::af_engp))
 298 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_audio_eng_t::io_count))
 299 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_audio_eng_t::intrate))

 301 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_audio_ctrl_t::statep))
 302 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_audio_ctrl_t::af_ctrlp))
 303 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_audio_ctrl_t::cval))

 305 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_ac_plumbed_t::acp_tqp))
 306 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_ac_plumbed_t::acp_uacp))

 308 _NOTE(DATA_READABLE_WITHOUT_LOCK(usb_audio_format_t::ch))

 310 /* usb_ac driver only care about two states:  plumbed or unplumbed */
 311 #define USB_AC_STATE_UNPLUMBED          0
 312 #define USB_AC_STATE_PLUMBED            1
 313 #define USB_AC_STATE_PLUMBED_RESTORING  2

 315 /* Default pipe states */
 316 #define USB_AC_DEF_CLOSED               0
 317 #define USB_AC_DEF_OPENED               1

 319 #define USB_AC_BUFFER_SIZE              256     /* descriptor buffer size */


 322 /*
 323  * delay before restoring state
 324  */
```



```
 325 #define USB_AC_RESTORE_DELAY            drv_usectohz(1000000)

 327 /* value for acp_driver */
 328 #define USB_AS_PLUMBED  1
 329 #define USB_AH_PLUMBED  2
 330 #define UNKNOWN_PLUMBED 3

 332 /* other useful macros */
 333 #if defined(__GNUC__)
 334 #define offsetof(s, m)  __builtin_offsetof(s, m)
 335 #else
 336 #endif /* ! codereview */
 337 #define offsetof(s, m)  ((size_t)(&(((s *)0)->m)))
 338 #endif
 339 #endif /* ! codereview */







 346 #define AF_REGISTERED   0x1
 347 #define AD_SETUP        0x10


 350 int usb_audio_attach(usb_ac_state_t *);
 351 /*
 352  * framework gain range
 353  */
 354 #define AUDIO_CTRL_STEREO_VAL(l, r)     (((l) & 0xff) | (((r) & 0xff) << 8))
 355 #define AUDIO_CTRL_STEREO_LEFT(v)       ((uint8_t)((v) & 0xff))
 356 #define AUDIO_CTRL_STEREO_RIGHT(v)      ((uint8_t)(((v) >> 8) & 0xff))


 359 #define AF_MAX_GAIN     100
 360 #define AF_MIN_GAIN     0



 364 int usb_ac_get_audio(void *, void *, int);

 366 void usb_ac_send_audio(void *, void *, int);

 368 void usb_ac_stop_play(usb_ac_state_t *, usb_audio_eng_t *);


 371 #ifdef __cplusplus
 372 }
 373 #endif

 375 #endif  /* _SYS_USB_AC_H */
```

```
**********************************************************
   11815 Thu Sep  5 23:02:18 2013
new/usr/src/uts/common/sys/usb/clients/video/usbvc/usbvc_var.h
3373 gcc >= 4.5 concerns about offsetof()
**********************************************************
______unchanged_portion_omitted_

 154 /* Macros */
 155 #define USBVC_OPEN              0x00000001

 157 /* For serialization. */
 158 #define USBVC_SER_NOSIG B_FALSE
 159 #define USBVC_SER_SIG           B_TRUE

 161 /*
 162  * Masks for debug printing
 163  */
 164 #define PRINT_MASK_ATTA         0x00000001
 165 #define PRINT_MASK_OPEN         0x00000002
 166 #define PRINT_MASK_CLOSE        0x00000004
 167 #define PRINT_MASK_READ         0x00000008
 168 #define PRINT_MASK_IOCTL        0x00000010
 169 #define PRINT_MASK_PM   0x00000020
 170 #define PRINT_MASK_CB   0x00000040
 171 #define PRINT_MASK_HOTPLUG      0x00000080
 172 #define PRINT_MASK_DEVCTRL      0x00000100
 173 #define PRINT_MASK_DEVMAP       0x00000200
 174 #define PRINT_MASK_ALL          0xFFFFFFFF

 176 #if defined(__GNUC__)
 177 #define offsetof(s, m)  __builtin_offsetof(s, m)
 178 #else
 179 #endif /* ! codereview */
 180 #define offsetof(s, m)  ((size_t)(&(((s *)0)->m)))
 181 #endif
 182 #endif /* ! codereview */

 184 #define USBVC_MAX_PKTS 40

 186 #define USBVC_DEFAULT_READ_BUF_NUM 3
 187 #define USBVC_MAX_READ_BUF_NUM 40
 188 #define USBVC_MAX_MAP_BUF_NUM 40

 190 /* According to UVC specs, the frame interval is in 100ns unit */
 191 #define USBVC_FRAME_INTERVAL_DENOMINATOR        10000000

 193 /* Only D3...D0 are writable, Table 4-6, UVC Spec */
 194 #define USBVC_POWER_MODE_MASK   0xf0;

 196 enum usbvc_buf_status {
 197         USBVC_BUF_INIT          = 0,  /* Allocated, to be queued */
 198             USBVC_BUF_MAPPED    = 1,  /* For map I/O only. Memory is mapped. */
 199             USBVC_BUF_EMPTY             = 2, /* not initialized, to be filled */

 201         /*
 202          * buf is filled with a full frame without any errors,
 203          * it will be moved to full list.
 204          */
 205             USBVC_BUF_DONE              = 4,

 207         /*
 208          * buf is filled to full but no EOF bit is found at the end
 209          * of video data
 210          */
 211             USBVC_BUF_ERR               = 8
 212 };
```



```
 214 /*
 215  * This structure is used to map v4l2 controls to uvc controls. The structure
 216  * array is addressed by (V4L2_CID_BASE - V4L2_CID_*)
 217  */
 218 typedef struct usbvc_v4l2_ctrl_map {
 219         char    name[32];
 220         uint8_t selector; /* Control Selector */
 221         uint8_t len;    /* wLength, defined in uvc spec chp 4 for each ctrl */

 223         /* The xth bit in bmControls bitmap of processing unit descriptor */
 224         uint8_t bit;

 226         enum    v4l2_ctrl_type type;
 227 } usbvc_v4l2_ctrl_map_t;

 229 typedef struct usbvc_v4l2_ctrl {
 230         uint8_t                 entity_id;
 231         usbvc_v4l2_ctrl_map_t   *ctrl_map;
 232 } usbvc_v4l2_ctrl_t;


 235 /*
 236  * State structure
 237  */
 238 struct usbvc_state {
 239         dev_info_t              *usbvc_dip;     /* per-device info handle */
 240         usb_client_dev_data_t   *usbvc_reg;     /* registration data */
 241         int                     usbvc_dev_state; /* USB device states. */
 242         int                     usbvc_drv_state; /* driver states. */
 243         kmutex_t                usbvc_mutex;
 244         kcondvar_t              usbvc_serial_cv;
 245         boolean_t               usbvc_serial_inuse;
 246         boolean_t               usbvc_locks_initialized;

 248         usbvc_power_t           *usbvc_pm;

 250         usb_log_handle_t        usbvc_log_handle;       /* log handle */
 251         usb_pipe_handle_t       usbvc_default_ph; /* default pipe */

 253         /* Video ctrl interface header descriptor */
 254         usbvc_vc_header_t       *usbvc_vc_header;
 255         list_t                  usbvc_term_list;
 256         list_t                  usbvc_unit_list;

 258         list_t                  usbvc_stream_list;
 259         usbvc_stream_if_t       *usbvc_curr_strm;
 260         kcondvar_t              usbvc_read_cv;  /* wait for read buf done */
 261         kcondvar_t              usbvc_mapio_cv; /* wait for mmap I/O buf done */

 263         /* current I/O type: read or mmap. */
 264         uchar_t                 usbvc_io_type;
 265 };


 268 /*
 269  * Used in ioctl entry to copy an argument from kernel space (arg_name)
 270  * to USER space (arg)
 271  */
 272 #define USBVC_COPYOUT(arg_name) \
 273 if (ddi_copyout(&arg_name, (caddr_t)arg, sizeof (arg_name), mode)) { \
 274     rv = EFAULT; \
 275     break;      \
 276 }

 278 /*
```

```
 * Used in ioctl entry to copy an argument from USER space (arg) to
 * KERNEL space (arg_name)
 */
#define USBVC_COPYIN(arg_name) \
if (ddi_copyin((caddr_t)arg, &arg_name, sizeof (arg_name), mode)) { \
        rv = EFAULT; \
        break;  \
}

/* Turn a little endian byte array to a uint32_t */
#define LE_TO_UINT32(src, off, des)     { \
                                uint32_t tmp; \
                                des = src[off + 3]; \
                                des = des << 24; \
                                tmp = src[off + 2]; \
                                des |= tmp << 16; \
                                tmp = src[off + 1]; \
                                des |= tmp << 8; \
                                des |= src[off]; \
                                }

/* Turn a uint32_t to a little endian byte array */
#define UINT32_TO_LE(src, off, des)     { \
                                des[off + 0] = 0xff & src; \
                                des[off + 1] = 0xff & (src >> 8); \
                                des[off + 2] = 0xff & (src >> 16); \
                                des[off + 3] = 0xff & (src >> 24); \
                                }

/* Turn a little endian byte array to a uint16_t */
#define LE_TO_UINT16(src, off, des)      \
                                des = src[off + 1]; \
                                des = des << 8; \
                                des |= src[off];

/* Turn a uint16_t to alittle endian byte array */
#define UINT16_TO_LE(src, off, des)     { \
                                des[off + 0] = 0xff & src; \
                                des[off + 1] = 0xff & (src >> 8); \
                                }

#define NELEM(a)        (sizeof (a) / sizeof (*(a)))

/* Minimum length of class specific descriptors */
#define USBVC_C_HEAD_LEN_MIN    12      /* ctrl header */
#define USBVC_I_TERM_LEN_MIN    8       /* input term */
#define USBVC_O_TERM_LEN_MIN    9       /* output term */
#define USBVC_P_UNIT_LEN_MIN    8       /* processing unit */
#define USBVC_S_UNIT_LEN_MIN    5       /* selector unit */
#define USBVC_E_UNIT_LEN_MIN    22      /* extension unit */
#define USBVC_FRAME_LEN_MIN     26      /* Frame descriptor */

/* Length of the Frame descriptor which has continuous frame intervals */
#define USBVC_FRAME_LEN_CON     38


/*
 * According to usb2.0 spec (table 9-13), for all ep, bits 10..0 specify the
 * max pkt size; for high speed ep, bits 12..11 specify the number of
 * additional transaction opportunities per microframe.
 */
#define HS_PKT_SIZE(pktsize) (pktsize & 0x07ff) * (1 + ((pktsize >> 11) & 3))

/*
 * warlock directives
 * _NOTE is an advice for locklint.  Locklint checks lock use for deadlocks.
```



```
 */
_NOTE(MUTEX_PROTECTS_DATA(usbvc_state_t::usbvc_mutex, usbvc_state_t))
_NOTE(DATA_READABLE_WITHOUT_LOCK(usbvc_state_t::{
        usbvc_dip
        usbvc_pm
        usbvc_log_handle
        usbvc_reg
        usbvc_default_ph
        usbvc_vc_header
        usbvc_term_list
        usbvc_unit_list
        usbvc_stream_list
}))

_NOTE(SCHEME_PROTECTS_DATA("stable data", usb_pipe_policy))
_NOTE(SCHEME_PROTECTS_DATA("USBA", usbvc_stream_if::datain_ph))
_NOTE(SCHEME_PROTECTS_DATA("USBA", usbvc_stream_if::curr_alt))
_NOTE(SCHEME_PROTECTS_DATA("USBA", usbvc_stream_if::curr_ep))
_NOTE(SCHEME_PROTECTS_DATA("unshared data", usbvc_buf::umem_cookie))
_NOTE(SCHEME_PROTECTS_DATA("unshared data", usbvc_buf::data))
_NOTE(SCHEME_PROTECTS_DATA("unshared data", usbvc_v4l2_ctrl))
_NOTE(SCHEME_PROTECTS_DATA("unshared data", usbvc_v4l2_ctrl_map))
_NOTE(SCHEME_PROTECTS_DATA("unshared data", mblk_t))
_NOTE(SCHEME_PROTECTS_DATA("unshared data", buf))
_NOTE(SCHEME_PROTECTS_DATA("unshared data", usb_isoc_req))
_NOTE(SCHEME_PROTECTS_DATA("unshared data", v4l2_queryctrl))
_NOTE(SCHEME_PROTECTS_DATA("unshared data", v4l2_format))
_NOTE(SCHEME_PROTECTS_DATA("unshared data", v4l2_control))
_NOTE(SCHEME_PROTECTS_DATA("unshared data", v4l2_streamparm))

int     usbvc_open_isoc_pipe(usbvc_state_t *, usbvc_stream_if_t *);
int     usbvc_start_isoc_polling(usbvc_state_t *, usbvc_stream_if_t *, uchar_t);
int     usbvc_vc_set_ctrl(usbvc_state_t *, uint8_t,  uint8_t,
                uint16_t, uint16_t, mblk_t *);
int     usbvc_vc_get_ctrl(usbvc_state_t *, uint8_t,  uint8_t,
                uint16_t, uint16_t, mblk_t *);
int     usbvc_vs_set_probe_commit(usbvc_state_t *, usbvc_stream_if_t *,
        usbvc_vs_probe_commit_t *, uchar_t);
void    usbvc_free_map_bufs(usbvc_state_t *, usbvc_stream_if_t *);
int     usbvc_alloc_map_bufs(usbvc_state_t *, usbvc_stream_if_t *, int, int);
int     usbvc_vs_get_probe(usbvc_state_t *, usbvc_stream_if_t *,
                usbvc_vs_probe_commit_t *, uchar_t);

/* Functions specific for V4L2 API */
uint8_t         usbvc_v4l2_colorspace(uint8_t);
uint32_t        usbvc_v4l2_guid2fcc(uint8_t *);
int             usbvc_v4l2_ioctl(usbvc_state_t *, int, intptr_t, int);


#ifdef __cplusplus
}
#endif

#endif  /* _SYS_USB_USBVC_VAR_H */
```

```
**********************************************************
    9250 Thu Sep  5 23:02:19 2013
new/usr/src/uts/common/sys/uwb/uwba/uwba.h
3373 gcc >= 4.5 concerns about offsetof()
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_UWB_UWBA_H
#define _SYS_UWB_UWBA_H

#ifdef  __cplusplus
extern "C" {
#endif


/*
 * UWBA private header file.
 */

#include <sys/note.h>
#include <sys/sunddi.h>
#include <sys/types.h>
#include <sys/list.h>
#include <sys/bitset.h>
#include <sys/bitmap.h>

#include <sys/uwb/uwb.h>
#include <sys/uwb/uwbai.h>

/* For logging. */
#define UWBA_LOG_DEBUG          2
#define UWBA_LOG_LOG            1
#define UWBA_LOG_CONSOLE        0

#if defined(__GNUC__)
#define offsetof(s, m)  __builtin_offsetof(s, m)
#else
#endif /* ! codereview */
#define offsetof(s, m)  ((size_t)(&(((s *)0)->m)))
#endif
#endif /* ! codereview */
#define isdigit(ch) ((ch >= '0') && (ch <= '9'))
```



```
#define UWB_RAW_RESULT_CODE_SIZE        5  /* size of RCEB + bResultCode */
#define UWB_RAW_RCCB_HEAD_SIZE          4  /* size of RCCB */

#define UWB_RAW_BEVENTTYPE_OFFSET       0  /* offset of bEventType */
#define UWB_RAW_WEVENT_OFFSET           1  /* offset of wEvent */
#define UWB_RAW_BEVENTCONTEXT_OFFSET    3  /* offset of bEventContext */
#define UWB_RAW_BRESULTCODE_OFFSET      4  /* offset of bResultCode */



#define UWB_CTXT_ID_TOP         0xfe    /* top context id */
#define UWB_CTXT_ID_BOTTOM      0x1     /* bottom context id */
#define UWB_CTXT_ID_NOTIF       0x0     /* notification context id */
#define UWB_CTXT_ID_UNVALID     0xff    /* invalid context id */


#define UWB_INVALID_EVT_CODE 0x7ffe     /* invalid evt/notif code */
#define UWB_INVALID_EVT_SIZE 0x7fff     /* invalid evt length */

#define UWB_MAX_NOTIF_NUMBER 10         /* Max notifications in a notif_list */

#define UWB_MAX_CDEV_NUMBER 32          /* Max client radio device */

/*
 * Offset of data rates Bits in PHY Capability Bitmap.
 * [ECMA, 16.8.16, table 112]
 */
#define UWB_RATE_OFFSET_BASE 16
/* the offset of data rate 53.3Mbps in PHY capability bitmap */
#define UWB_RATE_OFFSET_53 UWB_RATE_OFFSET_BASE
#define UWB_RATE_OFFSET_80 (UWB_RATE_OFFSET_BASE + 1) /* 80Mbps */
#define UWB_RATE_OFFSET_106 (UWB_RATE_OFFSET_BASE + 2)
#define UWB_RATE_OFFSET_160 (UWB_RATE_OFFSET_BASE + 3)
#define UWB_RATE_OFFSET_200 (UWB_RATE_OFFSET_BASE + 4)
#define UWB_RATE_OFFSET_320 (UWB_RATE_OFFSET_BASE + 5)
#define UWB_RATE_OFFSET_400 (UWB_RATE_OFFSET_BASE + 6)
#define UWB_RATE_OFFSET_480 (UWB_RATE_OFFSET_BASE + 7)

typedef int  (*uwb_rccb_handler_t)(uwb_dev_handle_t, uwb_rccb_cmd_t *);
#define UWB_RCCB_NULL_HANDLER ((uwb_rccb_handler_t)0)

#define UWB_STATE_IDLE          0
#define UWB_STATE_BEACON        1
#define UWB_STATE_SCAN          2

/* radio client device */
typedef struct uwba_client_dev {
        uint8_t                 bChannelNumber;
        uint8_t                 bBeaconType;
        uint16_t                wBPSTOffset;
        uwb_beacon_frame_t      beacon_frame;
        list_node_t             dev_node;
} uwba_client_dev_t;

/* Command result from the radio controller */
typedef struct uwb_cmd_result {
        uwb_rceb_head_t rceb;

        /* Cmd result data from device when cmd is finished. */
        uint8_t         buf[1];
} uwb_cmd_result_t;


typedef struct uwb_cmd_result_wrapper {
        /* Length of a uwb cmd_result */
        int                     length;
```

```
129         uwb_cmd_result_t        *cmd_result;
130 } uwb_cmd_result_wrapper_t;

132 typedef struct uwb_notif_wrapper {
133         /* Length of uwb notifcation */
134         int             length;
135         uwb_rceb_notif_t        *notif;

137         list_node_t     notif_node;
138 } uwb_notif_wrapper_t;



142 typedef struct uwba_dev {
143         /* dip of the uwb radio controller device */
144         dev_info_t      *dip;

146         /* Dev and instance */
147         char            *devinst;

149         kmutex_t        dev_mutex;

151         /* send cmd to the device */
152         int     (*send_cmd)(uwb_dev_handle_t, mblk_t *, uint16_t);

154         /* current command block */
155         uwb_rccb_cmd_t  curr_rccb;

157         /* wait for cmd complete and the cmd result available */
158         kcondvar_t      cmd_result_cv;
159         kcondvar_t      cmd_handler_cv;

161         /* filled by uwb_fill_cmd_result in rc driver's cmd call back */
162         uwb_cmd_result_wrapper_t cmd_result_wrap;

164         /*
165          * set to TRUE when start to do cmd ioctl;
166          * set to FALSE when put_cmd and exit cmd ioctl
167          */
168         boolean_t       cmd_busy;

170         /* Device state */
171         uint8_t         dev_state;

173         /* Beacon or scan channel */
174         uint8_t         channel;

176         /* Device address */
177         uint16_t        dev_addr;

179         /* notifications from radio controller device */
180         list_t          notif_list;

182         /* the current number of notifications in the notif_list */
183         int             notif_cnt;

185         /* client radio devices found through beacons by this radio host */
186         list_t          client_dev_list;

188         /* the current number of devices in dev_list */
189         int             client_dev_cnt;

191         /* context id is maintained by uwba */
192         uint8_t         ctxt_id;        /* current command context id */
193         bitset_t        ctxt_bits;      /* command context bit map */
```



```
195         /* PHY capability bitmap, saved from PHY capability IE */
196         ulong_t         phy_cap_bm;

198         /* list node of a uwb radio host device */
199         list_node_t     uwba_dev_node;
200 } uwba_dev_t;

202 _NOTE(MUTEX_PROTECTS_DATA(uwba_dev_t::dev_mutex, uwba_dev_t))
203 _NOTE(DATA_READABLE_WITHOUT_LOCK(uwba_dev_t::{
204         dip
205         devinst
206         send_cmd
207         phy_cap_bm
208         notif_cnt
209         dev_state
210         dip
211         ctxt_id
212         ctxt_bits
213         notif_list
214         cmd_result_wrap
215         client_dev_cnt
216         channel
217         dev_addr
218 }))


221 typedef struct uwba_evt_size {
222         /* length of a evt/notif structure, impact by alignment */
223         uint8_t struct_len;

225         /*
226          * offset of the length member of an event/notif struct.
227          * if zero, means there is no variable buf length member
228          * in this struct
229          */
230         uint16_t        buf_len_offset;
231 } uwba_evt_size_t;
232 typedef struct uwba_channel_range {
233         /* First channel in the specific bandgroup */
234         uint8_t base;

236         /* Length since this first channel in the bandgroup */
237         uint8_t offset;
238 }  uwba_channel_range_t;

240 #define UWB_RESULT_CODE_SIZE    (sizeof (uwb_rceb_result_code_t))

242 /* str_t is the struct type of the notif/evt */
243 #define UWB_EVT_RCEB_SZ         (sizeof (uwb_rceb_t))

245 /* the size after excluded the rceb head */
246 #define UWB_EVT_END_SZ(stru_t)  (sizeof (stru_t) - sizeof (uwb_rceb_t))

248 #define UWB_EVT_NO_BUF_LEN_OFFSET       0

250 /* Offset of wBeaconInfoLength in uwb_rceb_beacon_t */
251 #define UWB_BEACONINFOLEN_OFFSET 10

253 /* Offset of BeaconInfo from bChannelNumber in uwb_rceb_beacon_t */
254 #define UWB_BEACONINFO_OFFSET 8

256 /*
257  * UWB radio controller device list
258  */
259 void    uwba_dev_add_to_list(uwba_dev_t *);
```

```
260 void    uwba_dev_rm_from_list(uwba_dev_t *);
261 void    uwba_alloc_uwb_dev(dev_info_t *, uwba_dev_t **, uint_t);
262 void    uwba_free_uwb_dev(uwba_dev_t *);
263 uwb_dev_handle_t uwba_dev_search(dev_info_t *);

265 /*
266  * Context ID operations
267  */
268 void    uwba_init_ctxt_id(uwba_dev_t *);
269 void    uwba_fini_ctxt_id(uwba_dev_t *);
270 uint8_t uwba_get_ctxt_id(uwba_dev_t *);
271 void    uwba_free_ctxt_id(uwba_dev_t *, uint8_t);

273 void            uwba_fill_rccb_head(uwba_dev_t *, uint16_t, mblk_t *);
274 uint16_t        uwba_get_evt_code(uint8_t *, int);
275 uint16_t        uwba_get_evt_size(uint8_t *, int, uint16_t);

277 void    uwba_put_cmd_result(uwba_dev_t *, void *, uint16_t);
278 int     uwba_add_notif_to_list(uwba_dev_t *, void *, uint16_t);

280 /*
281  * Parse events/notifications from radio controller device
282  */
283 int     uwba_parse_data(char *, uchar_t *, size_t, void *, size_t);
284 int     uwba_parse_rceb(uint8_t *, size_t,      void *, size_t);
285 int     uwba_parse_dev_addr_mgmt(uint8_t *, int, uwb_rceb_dev_addr_mgmt_t *);
286 int     uwba_parse_get_ie(uwb_dev_handle_t, uint8_t *,
287         int, uwb_rceb_get_ie_t *);
288 int     uwba_parse_beacon_rcv(uwb_dev_handle_t, uint8_t *,
289         int, uwb_rceb_beacon_t *);
290 int     uwba_parse_bpoie_chg(uwb_dev_handle_t, uint8_t *,
291         int, uwb_rceb_bpoie_change_t *);
292 uint8_t uwba_allocate_channel(uwb_dev_handle_t);
293 uint8_t *uwba_find_ie(uwb_dev_handle_t, uint_t, uint8_t *, uint16_t);

295 void uwba_copy_rccb(uwb_rccb_cmd_t *, uwb_rccb_cmd_t *);

297 uwba_client_dev_t *uwba_find_cdev_by_channel(uwba_dev_t *, uint8_t);

299 /* Debug/message log */
300 void    uwba_log(uwba_dev_t *, uint_t, char *, ...);
301 const char *uwba_event_msg(uint16_t);

303 /* Turn a little endian byte array to a uint32_t */
304 #define LE_TO_UINT32(src, off, des) \
305 { \
306         uint32_t tmp; \
307         des = src[off + 3]; \
308         des = des << 24; \
309         tmp = src[off + 2]; \
310         des |= tmp << 16; \
311         tmp = src[off + 1]; \
312         des |= tmp << 8; \
313         des |= src[off]; \
314 }

316 /* Turn a uint32_t to a little endian byte array */
317 #define UINT32_TO_LE(src, off, des) \
318 { \
319         des[off + 0] = 0xff & src; \
320         des[off + 1] = 0xff & (src >> 8); \
321         des[off + 2] = 0xff & (src >> 16); \
322         des[off + 3] = 0xff & (src >> 24); \
323 }

325 /* Turn a little endian byte array to a uint16_t */
```



```
326 #define LE_TO_UINT16(src, off, des) \
327 { \
328         des = src[off + 1]; \
329         des = des << 8; \
330         des |= src[off]; \
331 }

333 /* Turn a uint16_t to alittle endian byte array */
334 #define UINT16_TO_LE(src, off, des) \
335 { \
336         des[off + 0] = 0xff & src; \
337         des[off + 1] = 0xff & (src >> 8); \
338 }


341 /* Max string length for the driver name and instance number. */
342 #define UWB_MAXSTRINGLEN 255


345 #ifdef __cplusplus
346 }
347 #endif

349 #endif  /* _SYS_UWB_UWBA_H */
```

```
**********************************************************
   50405 Thu Sep  5 23:02:20 2013
new/usr/src/uts/common/xen/io/xnb.c
3373 update files for xen
**********************************************************
______unchanged_portion_omitted_

1332 /*
1333  * Take packets from the peer and deliver them onward.
1334  */
1335 static mblk_t *
1336 xnb_from_peer(xnb_t *xnbp)
1337 {
1338         RING_IDX start, end, loop;
1339         gnttab_copy_t *cop;
1340         xnb_txbuf_t **txpp;
1341         netif_tx_request_t *txreq;
1342         boolean_t work_to_do, need_notify = B_FALSE;
1343         mblk_t *head, *tail;
1344         int n_data_req, i;

1346         ASSERT(MUTEX_HELD(&xnbp->xnb_tx_lock));

1348         head = tail = NULL;
1349 around:

1351         /* LINTED: constant in conditional context */
1352         RING_FINAL_CHECK_FOR_REQUESTS(&xnbp->xnb_tx_ring, work_to_do);
1353         if (!work_to_do) {
1354 finished:
1355                 xnb_tx_notify_peer(xnbp, need_notify);

1357                 return (head);
1358         }

1360         start = xnbp->xnb_tx_ring.req_cons;
1361         end = xnbp->xnb_tx_ring.sring->req_prod;

1363         if ((end - start) > NET_TX_RING_SIZE) {
1364                 /*
1365                  * This usually indicates that the frontend driver is
1366                  * misbehaving, as it's not possible to have more than
1367                  * NET_TX_RING_SIZE ring elements in play at any one
1368                  * time.
1369                  *
1370                  * We reset the ring pointers to the state declared by
1371                  * the frontend and try to carry on.
1372                  */
1373                 cmn_err(CE_WARN, "xnb_from_peer: domain %d tried to give us %u "
1374                     "items in the ring, resetting and trying to recover.",
1375                     xnbp->xnb_peer, (end - start));

1377                 /* LINTED: constant in conditional context */
1378                 BACK_RING_ATTACH(&xnbp->xnb_tx_ring,
1379                     (struct netif_tx_sring *)xnbp->xnb_tx_ring_addr, PAGESIZE);
1379                     (netif_tx_sring_t *)xnbp->xnb_tx_ring_addr, PAGESIZE);

1381                 goto around;
1382         }

1384         loop = start;
1385         cop = xnbp->xnb_tx_cop;
1386         txpp = xnbp->xnb_tx_bufp;
1387         n_data_req = 0;

1389         while (loop < end) {
```



```
1390                 static const uint16_t acceptable_flags =
1391                     NETTXF_csum_blank |
1392                     NETTXF_data_validated |
1393                     NETTXF_extra_info;
1394                 uint16_t unexpected_flags;

1396                 txreq = RING_GET_REQUEST(&xnbp->xnb_tx_ring, loop);

1398                 unexpected_flags = txreq->flags & ~acceptable_flags;
1399                 if (unexpected_flags != 0) {
1400                         /*
1401                          * The peer used flag bits that we do not
1402                          * recognize.
1403                          */
1404                         cmn_err(CE_WARN, "xnb_from_peer: "
1405                             "unexpected flag bits (0x%x) from peer "
1406                             "in transmit request",
1407                             unexpected_flags);
1408                         xnbp->xnb_stat_tx_unexpected_flags++;

1410                         /* Mark this entry as failed. */
1411                         xnb_tx_mark_complete(xnbp, txreq->id, NETIF_RSP_ERROR);
1412                         need_notify = B_TRUE;

1414                 } else if (txreq->flags & NETTXF_extra_info) {
1415                         struct netif_extra_info *erp;
1416                         boolean_t status;

1418                         loop++; /* Consume another slot in the ring. */
1419                         ASSERT(loop <= end);

1421                         erp = (struct netif_extra_info *)
1422                             RING_GET_REQUEST(&xnbp->xnb_tx_ring, loop);

1424                         switch (erp->type) {
1425                         case XEN_NETIF_EXTRA_TYPE_MCAST_ADD:
1426                                 ASSERT(xnbp->xnb_multicast_control);
1427                                 status = xnbp->xnb_flavour->xf_mcast_add(xnbp,
1428                                     &erp->u.mcast.addr);
1429                                 break;
1430                         case XEN_NETIF_EXTRA_TYPE_MCAST_DEL:
1431                                 ASSERT(xnbp->xnb_multicast_control);
1432                                 status = xnbp->xnb_flavour->xf_mcast_del(xnbp,
1433                                     &erp->u.mcast.addr);
1434                                 break;
1435                         default:
1436                                 status = B_FALSE;
1437                                 cmn_err(CE_WARN, "xnb_from_peer: "
1438                                     "unknown extra type %d", erp->type);
1439                                 break;
1440                         }

1442                         xnb_tx_mark_complete(xnbp, txreq->id,
1443                             status ? NETIF_RSP_OKAY : NETIF_RSP_ERROR);
1444                         need_notify = B_TRUE;

1446                 } else if ((txreq->offset > PAGESIZE) ||
1447                     (txreq->offset + txreq->size > PAGESIZE)) {
1448                         /*
1449                          * Peer attempted to refer to data beyond the
1450                          * end of the granted page.
1451                          */
1452                         cmn_err(CE_WARN, "xnb_from_peer: "
1453                             "attempt to refer beyond the end of granted "
1454                             "page in txreq (offset %d, size %d).",
1455                             txreq->offset, txreq->size);
```

```
1456                        xnbp->xnb_stat_tx_overflow_page++;

1458                        /* Mark this entry as failed. */
1459                        xnb_tx_mark_complete(xnbp, txreq->id, NETIF_RSP_ERROR);
1460                        need_notify = B_TRUE;

1462                } else {
1463                        xnb_txbuf_t *txp;

1465                        txp = kmem_cache_alloc(xnbp->xnb_tx_buf_cache,
1466                            KM_NOSLEEP);
1467                        if (txp == NULL)
1468                                break;

1470                        txp->xt_mblk = desballoc((unsigned char *)txp->xt_buf,
1471                            txp->xt_buflen, 0, &txp->xt_free_rtn);
1472                        if (txp->xt_mblk == NULL) {
1473                                kmem_cache_free(xnbp->xnb_tx_buf_cache, txp);
1474                                break;
1475                        }

1477                        txp->xt_idx = loop;
1478                        txp->xt_id = txreq->id;

1480                        cop->source.u.ref = txreq->gref;
1481                        cop->source.domid = xnbp->xnb_peer;
1482                        cop->source.offset = txreq->offset;

1484                        cop->dest.u.gmfn = txp->xt_mfn;
1485                        cop->dest.domid = DOMID_SELF;
1486                        cop->dest.offset = 0;

1488                        cop->len = txreq->size;
1489                        cop->flags = GNTCOPY_source_gref;
1490                        cop->status = 0;

1492                        *txpp = txp;

1494                        txpp++;
1495                        cop++;
1496                        n_data_req++;

1498                        ASSERT(n_data_req <= NET_TX_RING_SIZE);
1499                }

1501                loop++;
1502        }

1504        xnbp->xnb_tx_ring.req_cons = loop;

1506        if (n_data_req == 0)
1507                goto around;

1509        if (HYPERVISOR_grant_table_op(GNTTABOP_copy,
1510            xnbp->xnb_tx_cop, n_data_req) != 0) {

1512                cmn_err(CE_WARN, "xnb_from_peer: copy operation failed");

1514                txpp = xnbp->xnb_tx_bufp;
1515                i = n_data_req;
1516                while (i > 0) {
1517                        kmem_cache_free(xnbp->xnb_tx_buf_cache, *txpp);
1518                        txpp++;
1519                        i--;
1520                }
```



```
1522                goto finished;
1523        }

1525        txpp = xnbp->xnb_tx_bufp;
1526        cop = xnbp->xnb_tx_cop;
1527        i = n_data_req;

1529        while (i > 0) {
1530                xnb_txbuf_t *txp = *txpp;

1532                txreq = RING_GET_REQUEST(&xnbp->xnb_tx_ring, txp->xt_idx);

1534                if (cop->status != 0) {
1535 #ifdef XNB_DEBUG
1536                        cmn_err(CE_WARN, "xnb_from_peer: "
1537                            "txpp 0x%p failed (%d)",
1538                            (void *)*txpp, cop->status);
1539 #endif /* XNB_DEBUG */
1540                        xnb_tx_mark_complete(xnbp, txp->xt_id, NETIF_RSP_ERROR);
1541                        freemsg(txp->xt_mblk);
1542                } else {
1543                        mblk_t *mp;

1545                        mp = txp->xt_mblk;
1546                        mp->b_rptr = mp->b_wptr = (unsigned char *)txp->xt_buf;
1547                        mp->b_wptr += txreq->size;
1548                        mp->b_next = NULL;

1550                        /*
1551                         * If there are checksum flags, process them
1552                         * appropriately.
1553                         */
1554                        if ((txreq->flags &
1555                            (NETTXF_csum_blank | NETTXF_data_validated))
1556                            != 0) {
1557                                mp = xnbp->xnb_flavour->xf_cksum_from_peer(xnbp,
1558                                    mp, txreq->flags);
1559                                xnbp->xnb_stat_tx_cksum_no_need++;

1561                                txp->xt_mblk = mp;
1562                        }

1564                        if (head == NULL) {
1565                                ASSERT(tail == NULL);
1566                                head = mp;
1567                        } else {
1568                                ASSERT(tail != NULL);
1569                                tail->b_next = mp;
1570                        }
1571                        tail = mp;

1573                        xnbp->xnb_stat_opackets++;
1574                        xnbp->xnb_stat_obytes += txreq->size;

1576                        xnb_tx_mark_complete(xnbp, txp->xt_id, NETIF_RSP_OKAY);
1577                }

1579                txpp++;
1580                cop++;
1581                i--;
1582        }

1584        goto around;
1585        /* NOTREACHED */
1586 }
```

_____***unchanged_portion_omitted_***

```
1747 static boolean_t
1748 xnb_connect_rings(dev_info_t *dip)
1749 {
1750         xnb_t *xnbp = ddi_get_driver_private(dip);
1751         struct gnttab_map_grant_ref map_op;

1753         /*
1754          * Cannot attempt to connect the rings if already connected.
1755          */
1756         ASSERT(!xnbp->xnb_connected);

1758         /*
1759          * 1. allocate a vaddr for the tx page, one for the rx page.
1760          * 2. call GNTTABOP_map_grant_ref to map the relevant pages
1761          *    into the allocated vaddr (one for tx, one for rx).
1762          * 3. call EVTCHNOP_bind_interdomain to have the event channel
1763          *    bound to this domain.
1764          * 4. associate the event channel with an interrupt.
1765          * 5. enable the interrupt.
1766          */

1768         /* 1.tx */
1769         xnbp->xnb_tx_ring_addr = vmem_xalloc(heap_arena, PAGESIZE, PAGESIZE,
1770             0, 0, 0, 0, VM_SLEEP);
1771         ASSERT(xnbp->xnb_tx_ring_addr != NULL);

1773         /* 2.tx */
1774         map_op.host_addr = (uint64_t)((long)xnbp->xnb_tx_ring_addr);
1775         map_op.flags = GNTMAP_host_map;
1776         map_op.ref = xnbp->xnb_tx_ring_ref;
1777         map_op.dom = xnbp->xnb_peer;
1778         hat_prepare_mapping(kas.a_hat, xnbp->xnb_tx_ring_addr, NULL);
1779         if (xen_map_gref(GNTTABOP_map_grant_ref, &map_op, 1, B_FALSE) != 0 ||
1780             map_op.status != 0) {
1781                 cmn_err(CE_WARN, "xnb_connect_rings: cannot map tx-ring page.");
1782                 goto fail;
1783         }
1784         xnbp->xnb_tx_ring_handle = map_op.handle;

1786         /* LINTED: constant in conditional context */
1787         BACK_RING_INIT(&xnbp->xnb_tx_ring,
1788             (struct netif_tx_sring *)xnbp->xnb_tx_ring_addr, PAGESIZE);
1788             (netif_tx_sring_t *)xnbp->xnb_tx_ring_addr, PAGESIZE);

1790         /* 1.rx */
1791         xnbp->xnb_rx_ring_addr = vmem_xalloc(heap_arena, PAGESIZE, PAGESIZE,
1792             0, 0, 0, 0, VM_SLEEP);
1793         ASSERT(xnbp->xnb_rx_ring_addr != NULL);

1795         /* 2.rx */
1796         map_op.host_addr = (uint64_t)((long)xnbp->xnb_rx_ring_addr);
1797         map_op.flags = GNTMAP_host_map;
1798         map_op.ref = xnbp->xnb_rx_ring_ref;
1799         map_op.dom = xnbp->xnb_peer;
1800         hat_prepare_mapping(kas.a_hat, xnbp->xnb_rx_ring_addr, NULL);
1801         if (xen_map_gref(GNTTABOP_map_grant_ref, &map_op, 1, B_FALSE) != 0 ||
1802             map_op.status != 0) {
1803                 cmn_err(CE_WARN, "xnb_connect_rings: cannot map rx-ring page.");
1804                 goto fail;
1805         }
1806         xnbp->xnb_rx_ring_handle = map_op.handle;

1808         /* LINTED: constant in conditional context */
1809         BACK_RING_INIT(&xnbp->xnb_rx_ring,
1810             (struct netif_rx_sring *)xnbp->xnb_rx_ring_addr, PAGESIZE);
```



```
1810             (netif_rx_sring_t *)xnbp->xnb_rx_ring_addr, PAGESIZE);

1812         /* 3 */
1813         if (xvdi_bind_evtchn(dip, xnbp->xnb_fe_evtchn) != DDI_SUCCESS) {
1814                 cmn_err(CE_WARN, "xnb_connect_rings: "
1815                     "cannot bind event channel %d", xnbp->xnb_evtchn);
1816                 xnbp->xnb_evtchn = INVALID_EVTCHN;
1817                 goto fail;
1818         }
1819         xnbp->xnb_evtchn = xvdi_get_evtchn(dip);

1821         /*
1822          * It would be good to set the state to XenbusStateConnected
1823          * here as well, but then what if ddi_add_intr() failed?
1824          * Changing the state in the store will be noticed by the peer
1825          * and cannot be "taken back".
1826          */
1827         mutex_enter(&xnbp->xnb_tx_lock);
1828         mutex_enter(&xnbp->xnb_rx_lock);

1830         xnbp->xnb_connected = B_TRUE;

1832         mutex_exit(&xnbp->xnb_rx_lock);
1833         mutex_exit(&xnbp->xnb_tx_lock);

1835         /* 4, 5 */
1836         if (ddi_add_intr(dip, 0, NULL, NULL, xnb_intr, (caddr_t)xnbp)
1837             != DDI_SUCCESS) {
1838                 cmn_err(CE_WARN, "xnb_connect_rings: cannot add interrupt");
1839                 goto fail;
1840         }
1841         xnbp->xnb_irq = B_TRUE;

1843         return (B_TRUE);

1845 fail:
1846         mutex_enter(&xnbp->xnb_tx_lock);
1847         mutex_enter(&xnbp->xnb_rx_lock);

1849         xnbp->xnb_connected = B_FALSE;

1851         mutex_exit(&xnbp->xnb_rx_lock);
1852         mutex_exit(&xnbp->xnb_tx_lock);

1854         return (B_FALSE);
1855 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    5932 Thu Sep  5 23:02:20 2013
new/usr/src/uts/common/xen/io/xnb.h
3373 update files for xen
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * xnb.h - definitions for Xen dom0 network driver
 */

#ifndef _SYS_XNB_H
#define _SYS_XNB_H

#include <sys/types.h>
#include <sys/sysmacros.h>
#endif /* ! codereview */
#include <sys/kstat.h>
#include <sys/stream.h>
#include <sys/ethernet.h>
#include <sys/hypervisor.h>
#include <xen/public/io/netif.h>

#ifdef __cplusplus
extern "C" {
#endif

#define NET_TX_RING_SIZE  __CONST_RING_SIZE(netif_tx, PAGESIZE)
#define NET_RX_RING_SIZE  __CONST_RING_SIZE(netif_rx, PAGESIZE)
#define NET_TX_RING_SIZE  __RING_SIZE((netif_tx_sring_t *)0, PAGESIZE)
#define NET_RX_RING_SIZE  __RING_SIZE((netif_rx_sring_t *)0, PAGESIZE)

#define XNBMAXPKT       1500            /* MTU size */

/* DEBUG flags */
#define XNBDDI          0x01
#define XNBTRACE        0x02
#define XNBSEND         0x04
#define XNBRECV         0x08
#define XNBINTR         0x10
#define XNBRING         0x20
#define XNBCKSUM        0x40

#define XNB_STATE_INIT  0x01
```



```
#define XNB_STATE_READY 0x02

typedef struct xnb xnb_t;

/*
 * The xnb module provides core inter-domain network protocol functionality.
 * It is connected to the rest of Solaris in two ways:
 * - as a GLDv3 driver (with xnbu),
 * - as a GLDv3 consumer (with xnbo).
 *
 * The different modes of operation are termed "flavours" and each
 * instance of an xnb based driver operates in one and only one mode.
 * The common xnb driver exports a set of functions to these drivers
 * (declarations at the foot of this file) and calls back into the
 * drivers via the xnb_flavour_t structure.
 */
typedef struct xnb_flavour {
        void            (*xf_from_peer)(xnb_t *, mblk_t *);
        boolean_t       (*xf_peer_connected)(xnb_t *);
        void            (*xf_peer_disconnected)(xnb_t *);
        boolean_t       (*xf_hotplug_connected)(xnb_t *);
        boolean_t       (*xf_start_connect)(xnb_t *);
        mblk_t          *(*xf_cksum_from_peer)(xnb_t *, mblk_t *, uint16_t);
        uint16_t        (*xf_cksum_to_peer)(xnb_t *, mblk_t *);
        boolean_t       (*xf_mcast_add)(xnb_t *, ether_addr_t *);
        boolean_t       (*xf_mcast_del)(xnb_t *, ether_addr_t *);
} xnb_flavour_t;
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   66780 Thu Sep  5 23:02:21 2013
new/usr/src/uts/common/xen/io/xnf.c
3373 update files for xen
**********************************************************
______unchanged_portion_omitted_

2151 /*
2152  *  xnf_alloc_dma_resources() -- initialize the drivers structures
2153  */
2154 static int
2155 xnf_alloc_dma_resources(xnf_t *xnfp)
2156 {
2157         dev_info_t              *devinfo = xnfp->xnf_devinfo;
2158         size_t                  len;
2159         ddi_dma_cookie_t        dma_cookie;
2160         uint_t                  ncookies;
2161         int                     rc;
2162         struct netif_tx_sring *txs;
2163         struct netif_rx_sring *rxs;
2162         caddr_t                 rptr;

2165         /*
2166          * The code below allocates all the DMA data structures that
2167          * need to be released when the driver is detached.
2168          *
2169          * Allocate page for the transmit descriptor ring.
2170          */
2171         if (ddi_dma_alloc_handle(devinfo, &ringbuf_dma_attr,
2172             DDI_DMA_SLEEP, 0, &xnfp->xnf_tx_ring_dma_handle) != DDI_SUCCESS)
2173                 goto alloc_error;

2175         if (ddi_dma_mem_alloc(xnfp->xnf_tx_ring_dma_handle,
2176             PAGESIZE, &accattr, DDI_DMA_CONSISTENT,
2177             DDI_DMA_SLEEP, 0, (caddr_t *)&txs, &len,
2176             DDI_DMA_SLEEP, 0, &rptr, &len,
2178             &xnfp->xnf_tx_ring_dma_acchandle) != DDI_SUCCESS) {
2179                 ddi_dma_free_handle(&xnfp->xnf_tx_ring_dma_handle);
2180                 xnfp->xnf_tx_ring_dma_handle = NULL;
2181                 goto alloc_error;
2182         }

2184         if ((rc = ddi_dma_addr_bind_handle(xnfp->xnf_tx_ring_dma_handle, NULL,
2185             (caddr_t)&txs, PAGESIZE, DDI_DMA_RDWR | DDI_DMA_CONSISTENT,
2184             rptr, PAGESIZE, DDI_DMA_RDWR | DDI_DMA_CONSISTENT,
2186             DDI_DMA_SLEEP, 0, &dma_cookie, &ncookies)) != DDI_DMA_MAPPED) {
2187                 ddi_dma_mem_free(&xnfp->xnf_tx_ring_dma_acchandle);
2188                 ddi_dma_free_handle(&xnfp->xnf_tx_ring_dma_handle);
2189                 xnfp->xnf_tx_ring_dma_handle = NULL;
2190                 xnfp->xnf_tx_ring_dma_acchandle = NULL;
2191                 if (rc == DDI_DMA_NORESOURCES)
2192                         goto alloc_error;
2193                 else
2194                         goto error;
2195         }

2197         ASSERT(ncookies == 1);
2198         bzero(txs, PAGESIZE);
2197         bzero(rptr, PAGESIZE);
2199         /* LINTED: constant in conditional context */
2200         SHARED_RING_INIT(txs);
2199         SHARED_RING_INIT((netif_tx_sring_t *)rptr);
2201         /* LINTED: constant in conditional context */
2202         FRONT_RING_INIT(&xnfp->xnf_tx_ring, txs, PAGESIZE);
2201         FRONT_RING_INIT(&xnfp->xnf_tx_ring, (netif_tx_sring_t *)rptr, PAGESIZE);
2203         xnfp->xnf_tx_ring_phys_addr = dma_cookie.dmac_laddress;
```



```
2205         /*
2206          * Allocate page for the receive descriptor ring.
2207          */
2208         if (ddi_dma_alloc_handle(devinfo, &ringbuf_dma_attr,
2209             DDI_DMA_SLEEP, 0, &xnfp->xnf_rx_ring_dma_handle) != DDI_SUCCESS)
2210                 goto alloc_error;

2212         if (ddi_dma_mem_alloc(xnfp->xnf_rx_ring_dma_handle,
2213             PAGESIZE, &accattr, DDI_DMA_CONSISTENT,
2214             DDI_DMA_SLEEP, 0, (caddr_t *)&rxs, &len,
2213             DDI_DMA_SLEEP, 0, &rptr, &len,
2215             &xnfp->xnf_rx_ring_dma_acchandle) != DDI_SUCCESS) {
2216                 ddi_dma_free_handle(&xnfp->xnf_rx_ring_dma_handle);
2217                 xnfp->xnf_rx_ring_dma_handle = NULL;
2218                 goto alloc_error;
2219         }

2221         if ((rc = ddi_dma_addr_bind_handle(xnfp->xnf_rx_ring_dma_handle, NULL,
2222             (caddr_t)&rxs, PAGESIZE, DDI_DMA_RDWR | DDI_DMA_CONSISTENT,
2221             rptr, PAGESIZE, DDI_DMA_RDWR | DDI_DMA_CONSISTENT,
2223             DDI_DMA_SLEEP, 0, &dma_cookie, &ncookies)) != DDI_DMA_MAPPED) {
2224                 ddi_dma_mem_free(&xnfp->xnf_rx_ring_dma_acchandle);
2225                 ddi_dma_free_handle(&xnfp->xnf_rx_ring_dma_handle);
2226                 xnfp->xnf_rx_ring_dma_handle = NULL;
2227                 xnfp->xnf_rx_ring_dma_acchandle = NULL;
2228                 if (rc == DDI_DMA_NORESOURCES)
2229                         goto alloc_error;
2230                 else
2231                         goto error;
2232         }

2234         ASSERT(ncookies == 1);
2235         bzero(rxs, PAGESIZE);
2234         bzero(rptr, PAGESIZE);
2236         /* LINTED: constant in conditional context */
2237         SHARED_RING_INIT(rxs);
2236         SHARED_RING_INIT((netif_rx_sring_t *)rptr);
2238         /* LINTED: constant in conditional context */
2239         FRONT_RING_INIT(&xnfp->xnf_rx_ring, rxs, PAGESIZE);
2238         FRONT_RING_INIT(&xnfp->xnf_rx_ring, (netif_rx_sring_t *)rptr, PAGESIZE);
2240         xnfp->xnf_rx_ring_phys_addr = dma_cookie.dmac_laddress;

2242         return (DDI_SUCCESS);

2244 alloc_error:
2245         cmn_err(CE_WARN, "xnf%d: could not allocate enough DMA memory",
2246             ddi_get_instance(xnfp->xnf_devinfo));
2247 error:
2248         xnf_release_dma_resources(xnfp);
2249         return (DDI_FAILURE);
2250 }
______unchanged_portion_omitted_
```

```
**********************************************************
    4750 Thu Sep  5 23:02:22 2013
new/usr/src/uts/common/xen/io/xnf.h
3373 update files for xen
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_XNF_H
#define _SYS_XNF_H

#ifdef __cplusplus
extern "C" {
#endif

#define NET_TX_RING_SIZE  __CONST_RING_SIZE(netif_tx, PAGESIZE)
#define NET_RX_RING_SIZE  __CONST_RING_SIZE(netif_rx, PAGESIZE)
#define NET_TX_RING_SIZE  __RING_SIZE((netif_tx_sring_t *)0, PAGESIZE)
#define NET_RX_RING_SIZE  __RING_SIZE((netif_rx_sring_t *)0, PAGESIZE)

#define XNF_MAXPKT      1500            /* MTU size */
#define XNF_FRAMESIZE   1514            /* frame size including MAC header */

/* DEBUG flags */
#define XNF_DEBUG_DDI           0x01
#define XNF_DEBUG_TRACE         0x02

/*
 * Information about each receive buffer and any transmit look-aside
 * buffers.
 */
typedef struct xnf_buf {
        frtn_t                  free_rtn;
        struct xnf              *xnfp;
        ddi_dma_handle_t        dma_handle;
        caddr_t                 buf;            /* DMA-able data buffer */
        paddr_t                 buf_phys;
        mfn_t                   buf_mfn;
        size_t                  len;
        struct xnf_buf          *next;  /* For linking into free list */
        ddi_acc_handle_t        acc_handle;
        grant_ref_t             grant_ref;      /* grant table reference */
        uint16_t                id;             /* buffer id */
```



```
        unsigned int            gen;
} xnf_buf_t;
```

*unchanged_portion_omitted_*

```
**********************************************************
  194743 Thu Sep  5 23:02:23 2013
new/usr/src/uts/sun/io/scsi/adapters/sf.c
3373 gcc >= 4.5 concerns about offsetof()
**********************************************************
```

```
   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  * Copyright (c) 2011 Bayard G. Bell. All rights reserved.
  25  */

  27 /*
  28  * sf - Solaris Fibre Channel driver
  29  *
  30  * This module implements some of the Fibre Channel FC-4 layer, converting
  31  * from FC frames to SCSI and back.  (Note: no sequence management is done
  32  * here, though.)
  33  */

  35 #if defined(lint) && !defined(DEBUG)
  36 #define DEBUG   1
  37 #endif

  39 /*
  40  * XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  41  * Need to use the ugly RAID LUN mappings in FCP Annex D
  42  * to prevent SCSA from barfing.  This *REALLY* needs to
  43  * be addressed by the standards committee.
  44  */
  45 #define RAID_LUNS       1

  47 #ifdef DEBUG
  48 static int sfdebug = 0;
  49 #include <sys/debug.h>

  51 #define SF_DEBUG(level, args) \
  52         if (sfdebug >= (level)) sf_log args
  53 #else
  54 #define SF_DEBUG(level, args)
  55 #endif

  57 static int sf_bus_config_debug = 0;

  59 /* Why do I have to do this? */
  60 #if defined(__GNUC__)
  61 #define offsetof(s, m)  __builtin_offsetof(s, m)
```



```
  62 #else
  63 #define offsetof(s, m)  ((size_t)(&(((s *)0)->m)))
  64 #endif
  60 #define offsetof(s, m)  (size_t)(&(((s *)0)->m))

  66 #include <sys/scsi/scsi.h>
  67 #include <sys/fc4/fcal.h>
  68 #include <sys/fc4/fcp.h>
  69 #include <sys/fc4/fcal_linkapp.h>
  70 #include <sys/socal_cq_defs.h>
  71 #include <sys/fc4/fcal_transport.h>
  72 #include <sys/fc4/fcio.h>
  73 #include <sys/scsi/adapters/sfvar.h>
  74 #include <sys/scsi/impl/scsi_reset_notify.h>
  75 #include <sys/stat.h>
  76 #include <sys/varargs.h>
  77 #include <sys/var.h>
  78 #include <sys/thread.h>
  79 #include <sys/proc.h>
  80 #include <sys/kstat.h>
  81 #include <sys/devctl.h>
  82 #include <sys/scsi/targets/ses.h>
  83 #include <sys/callb.h>

  85 static int sf_info(dev_info_t *, ddi_info_cmd_t, void *, void **);
  86 static int sf_attach(dev_info_t *, ddi_attach_cmd_t);
  87 static int sf_detach(dev_info_t *, ddi_detach_cmd_t);
  88 static void sf_softstate_unlink(struct sf *);
  89 static int sf_scsi_bus_config(dev_info_t *parent, uint_t flag,
  90     ddi_bus_config_op_t op, void *arg, dev_info_t **childp);
  91 static int sf_scsi_bus_unconfig(dev_info_t *parent, uint_t flag,
  92     ddi_bus_config_op_t op, void *arg);
  93 static int sf_scsi_tgt_init(dev_info_t *, dev_info_t *,
  94     scsi_hba_tran_t *, struct scsi_device *);
  95 static void sf_scsi_tgt_free(dev_info_t *, dev_info_t *,
  96     scsi_hba_tran_t *, struct scsi_device *);
  97 static int sf_pkt_alloc_extern(struct sf *, struct sf_pkt *,
  98     int, int, int);
  99 static void sf_pkt_destroy_extern(struct sf *, struct sf_pkt *);
 100 static struct scsi_pkt *sf_scsi_init_pkt(struct scsi_address *,
 101     struct scsi_pkt *, struct buf *, int, int, int, int, int (*)(), caddr_t);
 102 static void sf_scsi_destroy_pkt(struct scsi_address *, struct scsi_pkt *);
 103 static void sf_scsi_dmafree(struct scsi_address *, struct scsi_pkt *);
 104 static void sf_scsi_sync_pkt(struct scsi_address *, struct scsi_pkt *);
 105 static int sf_scsi_reset_notify(struct scsi_address *, int,
 106     void (*)(caddr_t), caddr_t);
 107 static int sf_scsi_get_name(struct scsi_device *, char *, int);
 108 static int sf_scsi_get_bus_addr(struct scsi_device *, char *, int);
 109 static int sf_add_cr_pool(struct sf *);
 110 static int sf_cr_alloc(struct sf *, struct sf_pkt *, int (*)());
 111 static void sf_cr_free(struct sf_cr_pool *, struct sf_pkt *);
 112 static void sf_crpool_free(struct sf *);
 113 static int sf_kmem_cache_constructor(void *, void *, int);
 114 static void sf_kmem_cache_destructor(void *, void *);
 115 static void sf_statec_callback(void *, int);
 116 static int sf_login(struct sf *, uchar_t, uchar_t, uint_t, int);
 117 static int sf_els_transport(struct sf *, struct sf_els_hdr *);
 118 static void sf_els_callback(struct fcal_packet *);
 119 static int sf_do_prli(struct sf *, struct sf_els_hdr *, struct la_els_logi *);
 120 static int sf_do_adisc(struct sf *, struct sf_els_hdr *);
 121 static int sf_do_reportlun(struct sf *, struct sf_els_hdr *,
 122     struct sf_target *);
 123 static void sf_reportlun_callback(struct fcal_packet *);
 124 static int sf_do_inquiry(struct sf *, struct sf_els_hdr *,
 125     struct sf_target *);
 126 static void sf_inq_callback(struct fcal_packet *);
```

```
127 static struct fcal_packet *sf_els_alloc(struct sf *, uchar_t, int, int,
128     int, caddr_t *, caddr_t *);
129 static void sf_els_free(struct fcal_packet *);
130 static struct sf_target *sf_create_target(struct sf *,
131     struct sf_els_hdr *, int, int64_t);
132 #ifdef RAID_LUNS
133 static struct sf_target *sf_lookup_target(struct sf *, uchar_t *, int);
134 #else
135 static struct sf_target *sf_lookup_target(struct sf *, uchar_t *, int64_t);
136 #endif
137 static void sf_finish_init(struct sf *, int);
138 static void sf_offline_target(struct sf *, struct sf_target *);
139 static void sf_create_devinfo(struct sf *, struct sf_target *, int);
140 static int sf_create_props(dev_info_t *, struct sf_target *, int);
141 static int sf_commoncap(struct scsi_address *, char *, int, int, int);
142 static int sf_getcap(struct scsi_address *, char *, int);
143 static int sf_setcap(struct scsi_address *, char *, int, int);
144 static int sf_abort(struct scsi_address *, struct scsi_pkt *);
145 static int sf_reset(struct scsi_address *, int);
146 static void sf_abort_all(struct sf *, struct sf_target *, int, int, int);
147 static int sf_start(struct scsi_address *, struct scsi_pkt *);
148 static int sf_start_internal(struct sf *, struct sf_pkt *);
149 static void sf_fill_ids(struct sf *, struct sf_pkt *, struct sf_target *);
150 static int sf_prepare_pkt(struct sf *, struct sf_pkt *, struct sf_target *);
151 static int sf_dopoll(struct sf *, struct sf_pkt *);
152 static void sf_cmd_callback(struct fcal_packet *);
153 static void sf_throttle(struct sf *);
154 static void sf_watch(void *);
155 static void sf_throttle_start(struct sf *);
156 static void sf_check_targets(struct sf *);
157 static void sf_check_reset_delay(void *);
158 static int sf_target_timeout(struct sf *, struct sf_pkt *);
159 static void sf_force_lip(struct sf *);
160 static void sf_unsol_els_callback(void *, soc_response_t *, caddr_t);
161 static struct sf_els_hdr *sf_els_timeout(struct sf *, struct sf_els_hdr *);
162 /*PRINTFLIKE3*/
163 static void sf_log(struct sf *, int, const char *, ...);
164 static int sf_kstat_update(kstat_t *, int);
165 static int sf_open(dev_t *, int, int, cred_t *);
166 static int sf_close(dev_t, int, int, cred_t *);
167 static int sf_ioctl(dev_t, int, intptr_t, int, cred_t *, int *);
168 static struct sf_target *sf_get_target_from_dip(struct sf *, dev_info_t *);
169 static int sf_bus_get_eventcookie(dev_info_t *, dev_info_t *, char *,
170     ddi_eventcookie_t *);
171 static int sf_bus_add_eventcall(dev_info_t *, dev_info_t *,
172     ddi_eventcookie_t, void (*)(), void *, ddi_callback_id_t *cb_id);
173 static int sf_bus_remove_eventcall(dev_info_t *devi, ddi_callback_id_t cb_id);
174 static int sf_bus_post_event(dev_info_t *, dev_info_t *,
175     ddi_eventcookie_t, void *);

177 static void sf_hp_daemon(void *);

179 /*
180  * this is required to be able to supply a control node
181  * where ioctls can be executed
182  */
183 struct cb_ops sf_cb_ops = {
184         sf_open,                        /* open */
185         sf_close,                       /* close */
186         nodev,                          /* strategy */
187         nodev,                          /* print */
188         nodev,                          /* dump */
189         nodev,                          /* read */
190         nodev,                          /* write */
191         sf_ioctl,                       /* ioctl */
192         nodev,                          /* devmap */
```



```
193         nodev,                          /* mmap */
194         nodev,                          /* segmap */
195         nochpoll,                       /* poll */
196         ddi_prop_op,                    /* cb_prop_op */
197         0,                              /* streamtab  */
198         D_MP | D_NEW | D_HOTPLUG        /* driver flags */

200 };
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   99457 Thu Sep  5 23:02:24 2013
new/usr/src/uts/sun4u/io/rmclomv.c
3373 gcc >= 4.5 concerns about offsetof()
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */


#include <sys/types.h>
#include <sys/stat.h>
#include <sys/conf.h>
#include <sys/modctl.h>
#include <sys/callb.h>
#include <sys/strlog.h>
#include <sys/cyclic.h>
#include <sys/rmc_comm_dp.h>
#include <sys/rmc_comm_dp_boot.h>
#include <sys/rmc_comm_drvintf.h>
#include <sys/rmc_comm.h>
#include <sys/machsystm.h>
#include <sys/sysevent.h>
#include <sys/sysevent/dr.h>
#include <sys/sysevent/env.h>
#include <sys/sysevent/eventdefs.h>
#include <sys/file.h>
#include <sys/disp.h>
#include <sys/reboot.h>
#include <sys/envmon.h>
#include <sys/rmclomv_impl.h>
#include <sys/cpu_sgnblk_defs.h>
#include <sys/utsname.h>
#include <sys/systeminfo.h>
#include <sys/ddi.h>
#include <sys/time.h>
#include <sys/promif.h>

#if defined(__GNUC__)
#define offsetof(s, m)  __builtin_offsetof(s, m)
#else
#define offsetof(s, m)  ((size_t)(&(((s *)0)->m)))
#endif
#define offsetof(s, m)  (size_t)(&(((s *)0)->m))
```



```
#define RMCRESBUFLEN    1024
#define DATE_TIME_MSG_SIZE      78
#define RMCLOMV_WATCHDOG_MODE   "rmclomv-watchdog-mode"
#define DELAY_TIME      5000000  /* 5 seconds, in microseconds */
#define CPU_SIGNATURE_DELAY_TIME        5000000  /* 5 secs, in microsecs */

extern void     pmugpio_watchdog_pat();

extern int      watchdog_activated;
static int      last_watchdog_msg = 1;
extern int      watchdog_enable;
extern int      boothowto;

int             rmclomv_watchdog_mode;

/*
 * functions local to this driver.
 */
static int      rmclomv_getinfo(dev_info_t *dip, ddi_info_cmd_t cmd, void *arg,
    void **resultp);
static int      rmclomv_attach(dev_info_t *dip, ddi_attach_cmd_t cmd);
static int      rmclomv_detach(dev_info_t *dip, ddi_detach_cmd_t cmd);
static uint_t   rmclomv_break_intr(caddr_t arg);
static int      rmclomv_add_intr_handlers(void);
static int      rmclomv_remove_intr_handlers(void);
static uint_t   rmclomv_event_data_handler(char *);
static void     rmclomv_dr_data_handler(const char *, int);
static int      rmclomv_open(dev_t *dev_p, int flag, int otyp, cred_t *cred_p);
static int      rmclomv_close(dev_t dev, int flag, int otyp, cred_t *cred_p);
static int      rmclomv_ioctl(dev_t dev, int cmd, intptr_t arg, int mode,
    cred_t *cred_p, int *rval_p);
static void     rmclomv_checkrmc_start(void);
static void     rmclomv_checkrmc_destroy(void);
static void     rmclomv_checkrmc_wakeup(void *);
static void     rmclomv_refresh_start(void);
static void     rmclomv_refresh_destroy(void);
static void     rmclomv_refresh_wakeup(void);
static void     rmclomv_reset_cache(rmclomv_cache_section_t *new_chain,
    rmclomv_cache_section_t *new_subchain, dp_get_sysinfo_r_t *sysinfo);
static rmclomv_cache_section_t *rmclomv_find_section(
    rmclomv_cache_section_t *start, uint16_t sensor);
static rmclomv_cache_section_t *create_cache_section(int sensor_type, int num);
static int      get_sensor_by_name(const rmclomv_cache_section_t *section,
    const char *name, int *index);
static int      validate_section_entry(rmclomv_cache_section_t *section,
    int index);
static int      add_names_to_section(rmclomv_cache_section_t *section);
static void     free_section(rmclomv_cache_section_t *section);
static void     add_section(rmclomv_cache_section_t **head,
    rmclomv_cache_section_t *section);
static int      rmclomv_do_cmd(int req_cmd, int resp_cmd, int resp_len,
    intptr_t arg_req, intptr_t arg_res);
static void     refresh_name_cache(int force_fail);
static void     set_val_unav(envmon_sensor_t *sensor);
static void     set_fan_unav(envmon_fan_t *fan);
static int      do_psu_cmd(intptr_t arg, int mode, envmon_indicator_t *env_ind,
    dp_get_psu_status_t *rmc_psu, dp_get_psu_status_r_t *rmc_psu_r,
    int detector_type);
static uint_t rmc_set_watchdog_timer(uint_t timeoutval);
static uint_t rmc_clear_watchdog_timer(void);
static void send_watchdog_msg(int msg);
static void plat_timesync(void *arg);

static kmutex_t         timesync_lock;
static clock_t          timesync_interval = 0;
static timeout_id_t     timesync_tid = 0;
```

```
128 /*
129  * Driver entry points
130  */
131 static struct cb_ops rmclomv_cb_ops = {
132         rmclomv_open,   /* open */
133         rmclomv_close,  /* close */
134         nodev,          /* strategy() */
135         nodev,          /* print() */
136         nodev,          /* dump() */
137         nodev,          /* read() */
138         nodev,          /* write() */
139         rmclomv_ioctl,  /* ioctl() */
140         nodev,          /* devmap() */
141         nodev,          /* mmap() */
142         ddi_segmap,     /* segmap() */
143         nochpoll,       /* poll() */
144         ddi_prop_op,    /* prop_op() */
145         NULL,           /* cb_str */
146         D_NEW | D_MP    /* cb_flag */
147 };
```

*unchanged_portion_omitted_*