```
**********************************************************
    1267 Thu Nov 22 14:24:26 2012
new/usr/src/cmd/stat/Makefile
749 "/usr/bin/kstat" should be rewritten in C
Reviewed by: Garrett D'Amore <garrett@damore.org>
Reviewed by: Brendan Gregg <brendan.gregg@joyent.com>
**********************************************************
```

```
   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 2006 Sun Microsystems, Inc.  All rights reserved.
  23 # Use is subject to license terms.
  24 #
  25 #ident  "%Z%%M% %I%     %E% SMI"
  26 #
  25 # cmd/stat/Makefile
  26 #

  28 include ../Makefile.cmd

  30 SUBDIRS=        iostat mpstat vmstat fsstat kstat
  32 SUBDIRS=        iostat mpstat vmstat fsstat

  32 all :=          TARGET = all
  33 install :=      TARGET = install
  34 clean :=        TARGET = clean
  35 clobber :=      TARGET = clobber
  36 lint :=         TARGET = lint
  37 _msg :=         TARGET = _msg

  39 .KEEP_STATE:

  41 all install lint clean clobber _msg: $(SUBDIRS)

  43 $(SUBDIRS): FRC
  44         @cd $@; pwd; $(MAKE) $(MFLAGS) $(TARGET)

  46 FRC:
```

```
**********************************************************
    1613 Thu Nov 22 14:24:26 2012
new/usr/src/cmd/stat/kstat/Makefile
749 "/usr/bin/kstat" should be rewritten in C
Reviewed by: Garrett D'Amore <garrett@damore.org>
Reviewed by: Brendan Gregg <brendan.gregg@joyent.com>
**********************************************************
```

```
#
# 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.
#

PROG = kstat
OBJS = kstat.o
SRCS =$(OBJS:%.o=%.c) $(COMMON_SRCS)

include $(SRC)/cmd/Makefile.cmd
include $(SRC)/cmd/stat/Makefile.stat

LDLIBS += -lavl -lcmdutils -ldevinfo -lgen -lkstat
CFLAGS += $(CCVERBOSE) -I${STATCOMMONDIR}
CERRWARN += -_gcc=-Wno-uninitialized
CERRWARN += -_gcc=-Wno-switch
CERRWARN += -_gcc=-Wno-parentheses
FILEMODE= 0555

lint := LINTFLAGS = -muxs -I$(STATCOMMONDIR)

.KEEP_STATE:

all: $(PROG)

install: all $(ROOTPROG)

$(PROG): $(OBJS) $(COMMON_OBJS)
        $(LINK.c) -o $(PROG) $(OBJS) $(COMMON_OBJS) $(LDLIBS)
        $(POST_PROCESS)

%.o : $(STATCOMMONDIR)/%.c
        $(COMPILE.c) -o $@ $<
        $(POST_PROCESS_O)

clean:
        -$(RM) $(OBJS) $(COMMON_OBJS)

lint: lint_SRCS
```



```
include $(SRC)/cmd/Makefile.targ
#endif /* ! codereview */
```

```
**********************************************************
   33906 Thu Nov 22 14:24:27 2012
new/usr/src/cmd/stat/kstat/kstat.c
749 "/usr/bin/kstat" should be rewritten in C
Reviewed by: Garrett D'Amore <garrett@damore.org>
Reviewed by: Brendan Gregg <brendan.gregg@joyent.com>
**********************************************************
```

```
/*
 * 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) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2012 David Hoeppner. All rights reserved.
 */

/*
 * Display kernel statistics
 *
 * This is a reimplementation of the perl kstat command originally found
 * under usr/src/cmd/kstat/kstat.pl
 *
 * Incompatibilities:
 *      - perl regular expressions not longer supported
 *      - options checking is stricter
 *
 * Flags added:
 *      -C      similar to the -p option but value is separated by a colon
 *      -h      display help
 *      -j      json format
 */

#include <assert.h>
#include <ctype.h>
#include <errno.h>
#include <kstat.h>
#include <langinfo.h>
#include <libgen.h>
#include <limits.h>
#include <locale.h>
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <time.h>
#include <unistd.h>
#include <sys/list.h>
#include <sys/time.h>
```



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

#include "kstat.h"
#include "statcommon.h"

char    *cmdname = "kstat";
int     caught_cont = 0;

static uint_t   g_timestamp_fmt = NODATE;

/* Helper flag - header was printed already? */
static boolean_t g_headerflg;

/* Saved command line options */
static boolean_t g_cflg = B_FALSE;
static boolean_t g_jflg = B_FALSE;
static boolean_t g_lflg = B_FALSE;
static boolean_t g_pflg = B_FALSE;
static boolean_t g_qflg = B_FALSE;
static char     *g_ks_class = "*";

/* Return zero if a selector did match */
static int      g_matched = 1;

/* Sorted list of kstat instances */
static list_t   instances_list;
static list_t   selector_list;

int
main(int argc, char **argv)
{
        ks_selector_t   *nselector;
        ks_selector_t   *uselector;
        kstat_ctl_t     *kc;
        hrtime_t        start_n;
        hrtime_t        period_n;
        boolean_t       errflg = B_FALSE;
        boolean_t       nselflg = B_FALSE;
        boolean_t       uselflg = B_FALSE;
        char            *q;
        int             count = 1;
        int             infinite_cycles = 0;
        int             interval = 0;
        int             n = 0;
        int             c, m, tmp;

        (void) setlocale(LC_ALL, "");
#if !defined(TEXT_DOMAIN)               /* Should be defined by cc -D */
#define TEXT_DOMAIN "SYS_TEST"          /* Use this only if it wasn't */
#endif
        (void) textdomain(TEXT_DOMAIN);

        /*
         * Create the selector list and a dummy default selector to match
         * everything. While we process the cmdline options we will add
         * selectors to this list.
         */
        list_create(&selector_list, sizeof (ks_selector_t),
            offsetof(ks_selector_t, ks_next));

        nselector = new_selector();

        /*
         * Parse named command line arguments.
         */
        while ((c = getopt(argc, argv, "h?CqjlpT:m:i:n:s:c:")) != EOF)
```

```
126                 switch (c) {
127                 case 'h':
128                 case '?':
129                         usage();
130                         exit(0);
131                         break;
132                 case 'C':
133                         g_pflg = g_cflg = B_TRUE;
134                         break;
135                 case 'q':
136                         g_qflg = B_TRUE;
137                         break;
138                 case 'j':
139                         g_jflg = B_TRUE;
140                         break;
141                 case 'l':
142                         g_pflg = g_lflg = B_TRUE;
143                         break;
144                 case 'p':
145                         g_pflg = B_TRUE;
146                         break;
147                 case 'T':
148                         switch (*optarg) {
149                         case 'd':
150                                 g_timestamp_fmt = DDATE;
151                                 break;
152                         case 'u':
153                                 g_timestamp_fmt = UDATE;
154                                 break;
155                         default:
156                                 errflg = B_TRUE;
157                         }
158                         break;
159                 case 'm':
160                         nselflg = B_TRUE;
161                         nselector->ks_module =
162                             (char *)safe_strdup(optarg);
163                         break;
164                 case 'i':
165                         nselflg = B_TRUE;
166                         nselector->ks_instance =
167                             (char *)safe_strdup(optarg);
168                         break;
169                 case 'n':
170                         nselflg = B_TRUE;
171                         nselector->ks_name =
172                             (char *)safe_strdup(optarg);
173                         break;
174                 case 's':
175                         nselflg = B_TRUE;
176                         nselector->ks_statistic =
177                             (char *)safe_strdup(optarg);
178                         break;
179                 case 'c':
180                         g_ks_class =
181                             (char *)safe_strdup(optarg);
182                         break;
183                 default:
184                         errflg = B_TRUE;
185                         break;
186                 }

188         if (g_qflg && (g_jflg || g_pflg)) {
189                 (void) fprintf(stderr, gettext(
190                     "-q and -lpj are mutually exclusive\n"));
191                 errflg = B_TRUE;
```



```
192         }

194         if (errflg) {
195                 usage();
196                 exit(2);
197         }

199         argc -= optind;
200         argv += optind;

202         /*
203          * Consume the rest of the command line. Parsing the
204          * unnamed command line arguments.
205          */
206         while (argc--) {
207                 errno = 0;
208                 tmp = strtoul(*argv, &q, 10);
209                 if (tmp == ULONG_MAX && errno == ERANGE) {
210                         if (n == 0) {
211                                 (void) fprintf(stderr, gettext(
212                                     "Interval is too large\n"));
213                         } else if (n == 1) {
214                                 (void) fprintf(stderr, gettext(
215                                     "Count is too large\n"));
216                         }
217                         usage();
218                         exit(2);
219                 }

221                 if (errno != 0 || *q != '\0') {
222                         m = 0;
223                         uselector = new_selector();
224                         while ((q = (char *)strsep(argv, ":")) != NULL) {
225                                 m++;
226                                 if (m > 4) {
227                                         free(uselector);
228                                         usage();
229                                         exit(2);
230                                 }

232                                 if (*q != '\0') {
233                                         switch (m) {
234                                         case 1:
235                                                 uselector->ks_module =
236                                                     (char *)safe_strdup(q);
237                                                 break;
238                                         case 2:
239                                                 uselector->ks_instance =
240                                                     (char *)safe_strdup(q);
241                                                 break;
242                                         case 3:
243                                                 uselector->ks_name =
244                                                     (char *)safe_strdup(q);
245                                                 break;
246                                         case 4:
247                                                 uselector->ks_statistic =
248                                                     (char *)safe_strdup(q);
249                                                 break;
250                                         default:
251                                                 assert(B_FALSE);
252                                         }
253                                 }
254                         }

256                         if (m < 4) {
257                                 free(uselector);
```

```
258                                 usage();
259                                 exit(2);
260                         }

262                         uselflg = B_TRUE;
263                         list_insert_tail(&selector_list, uselector);
264                 } else {
265                         if (tmp < 1) {
266                                 if (n == 0) {
267                                         (void) fprintf(stderr, gettext(
268                                             "interval must be an "
269                                             "integer >= 1"));
270                                 } else if (n == 1) {
271                                         (void) fprintf(stderr, gettext(
272                                             "Count must be an integer >= 1"));
273                                 }
274                                 usage();
275                                 exit(2);
276                         } else {
277                                 if (n == 0) {
278                                         interval = tmp;
279                                         count = -1;
280                                 } else if (n == 1) {
281                                         count = tmp;
282                                 } else {
283                                         usage();
284                                         exit(2);
285                                 }
286                         }
287                         n++;
288                 }
289                 argv++;
290         }

292         /*
293          * Check if we founded a named selector on the cmdline.
294          */
295         if (uselflg) {
296                 if (nselflg) {
297                         (void) fprintf(stderr, gettext(
298                             "module:instance:name:statistic and "
299                             "-m -i -n -s are mutually exclusive"));
300                         usage();
301                         exit(2);
302                 } else {
303                         free(nselector);
304                 }
305         } else {
306                 list_insert_tail(&selector_list, nselector);
307         }

309         assert(!list_is_empty(&selector_list));

311         list_create(&instances_list, sizeof (ks_instance_t),
312             offsetof(ks_instance_t, ks_next));

314         kc = kstat_open();
315         if (kc == NULL)  {
316                 perror("kstat_open");
317                 exit(3);
318         }

320         period_n = (hrtime_t)interval * NANOSEC;
321         start_n = gethrtime();

323         while (count == -1 || count-- > 0) {
```



```
324                 ks_instances_read(kc);
325                 ks_instances_print();

327                 if (interval && count) {
328                         sleep_until(&start_n, period_n, infinite_cycles,
329                             &caught_cont);
330                         (void) kstat_chain_update(kc);
331                         (void) putchar('\n');
332                 }
333         }

335         (void) kstat_close(kc);

337         return (g_matched);
338 }

340 /*
341  * Print usage.
342  */
343 static void
344 usage(void)
345 {
346         (void) fprintf(stderr, gettext(
347             "Usage:\n"
348             "kstat [ -Cjlpq ] [ -T d|u ] [ -c class ]\n"
349             "      [ -m module ] [ -i instance ] [ -n name ] [ -s statistic ]\n"
350             "      [ interval [ count ] ]\n"
351             "kstat [ -Cjlpq ] [ -T d|u ] [ -c class ]\n"
352             "      [ module:instance:name:statistic ... ]\n"
353             "      [ interval [ count ] ]\n"));
354 }

356 /*
357  * Sort compare function.
358  */
359 static int
360 compare_instances(ks_instance_t *l_arg, ks_instance_t *r_arg)
361 {
362         int     rval;

364         rval = strcasecmp(l_arg->ks_module, r_arg->ks_module);
365         if (rval == 0) {
366                 if (l_arg->ks_instance == r_arg->ks_instance) {
367                         return (strcasecmp(l_arg->ks_name, r_arg->ks_name));
368                 } else if (l_arg->ks_instance < r_arg->ks_instance) {
369                         return (-1);
370                 } else {
371                         return (1);
372                 }
373         } else {
374                 return (rval);
375         }
376 }

378 /*
379  * Inserts an instance in the per selector list.
380  */
381 static void
382 nvpair_insert(ks_instance_t *ksi, char *name, ks_value_t *value,
383     uchar_t data_type)
384 {
385         ks_nvpair_t     *instance;
386         ks_nvpair_t     *tmp;

388         instance = (ks_nvpair_t *)malloc(sizeof (ks_nvpair_t));
389         if (instance == NULL) {
```

```
390                 perror("malloc");
391                 exit(3);
392         }

394         (void) strlcpy(instance->name, name, KSTAT_STRLEN);
395         (void) memcpy(&instance->value, value, sizeof (ks_value_t));
396         instance->data_type = data_type;

398         tmp = list_head(&ksi->ks_nvlist);
399         while (tmp != NULL && strcasecmp(instance->name, tmp->name) > 0)
400                 tmp = list_next(&ksi->ks_nvlist, tmp);

402         list_insert_before(&ksi->ks_nvlist, tmp, instance);
403 }

405 /*
406  * Allocates a new all-matching selector.
407  */
408 static ks_selector_t *
409 new_selector(void)
410 {
411         ks_selector_t   *selector;

413         selector = (ks_selector_t *)malloc(sizeof (ks_selector_t));
414         if (selector == NULL) {
415                 perror("malloc");
416                 exit(3);
417         }

419         list_link_init(&selector->ks_next);

421         selector->ks_module = "*";
422         selector->ks_instance = "*";
423         selector->ks_name = "*";
424         selector->ks_statistic = "*";

426         return (selector);
427 }

429 /*
430  * This function was taken from the perl kstat module code - please
431  * see for further comments there.
432  */
433 static kstat_raw_reader_t
434 lookup_raw_kstat_fn(char *module, char *name)
435 {
436         char            key[KSTAT_STRLEN * 2];
437         register char   *f, *t;
438         int             n = 0;

440         for (f = module, t = key; *f != '\0'; f++, t++) {
441                 while (*f != '\0' && isdigit(*f))
442                         f++;
443                 *t = *f;
444         }
445         *t++ = ':';

447         for (f = name; *f != '\0'; f++, t++) {
448                 while (*f != '\0' && isdigit(*f))
449                         f++;
450                 *t = *f;
451         }
452         *t = '\0';

454         while (ks_raw_lookup[n].fn != NULL) {
455                 if (strncmp(ks_raw_lookup[n].name, key, strlen(key)) == 0)
```



```
456                         return (ks_raw_lookup[n].fn);
457                 n++;
458         }

460         return (0);
461 }

463 /*
464  * Iterate over all kernel statistics and save matches.
465  */
466 static void
467 ks_instances_read(kstat_ctl_t *kc)
468 {
469         kstat_raw_reader_t save_raw = NULL;
470         kid_t           id;
471         ks_selector_t   *selector;
472         ks_instance_t   *ksi;
473         ks_instance_t   *tmp;
474         kstat_t         *kp;
475         boolean_t       skip;
476         char            *ks_number;

478         for (kp = kc->kc_chain; kp != NULL; kp = kp->ks_next) {
479                 /* Don't bother storing the kstat headers */
480                 if (strncmp(kp->ks_name, "kstat_", 6) == 0) {
481                         continue;
482                 }

484                 /* Don't bother storing raw stats we don't understand */
485                 if (kp->ks_type == KSTAT_TYPE_RAW) {
486                         save_raw = lookup_raw_kstat_fn(kp->ks_module,
487                             kp->ks_name);
488                         if (save_raw == NULL) {
489 #ifdef REPORT_UNKNOWN
490                                 (void) fprintf(stderr,
491                                     "Unknown kstat type %s:%d:%s - "
492                                     "%d of size %d\n", kp->ks_module,
493                                     kp->ks_instance, kp->ks_name,
494                                     kp->ks_ndata, kp->ks_data_size);
495 #endif
496                                 continue;
497                         }
498                 }

500                 /*
501                  * Iterate over the list of selectors and skip
502                  * instances we dont want. We filter for statistics
503                  * later, as we dont know them yet.
504                  */
505                 skip = B_FALSE;
506                 (void) asprintf(&ks_number, "%d", kp->ks_instance);
507                 selector = list_head(&selector_list);
508                 while (selector != NULL) {
509                         if (!(gmatch(kp->ks_module, selector->ks_module) != 0 &&
510                             gmatch(ks_number, selector->ks_instance) != 0 &&
511                             gmatch(kp->ks_name, selector->ks_name) != 0 &&
512                             gmatch(kp->ks_class, g_ks_class))) {
513                                 skip = B_TRUE;
514                         }
515                         selector = list_next(&selector_list, selector);
516                 }

518                 free(ks_number);

520                 if (skip) {
521                         continue;
```

```
522                 }
523
524                 /*
525                  * Allocate a new instance and fill in the values
526                  * we know so far.
527                  */
528                 ksi = (ks_instance_t *)malloc(sizeof (ks_instance_t));
529                 if (ksi == NULL) {
530                         perror("malloc");
531                         exit(3);
532                 }
533
534                 list_link_init(&ksi->ks_next);
535
536                 (void) strlcpy(ksi->ks_module, kp->ks_module, KSTAT_STRLEN);
537                 (void) strlcpy(ksi->ks_name, kp->ks_name, KSTAT_STRLEN);
538                 (void) strlcpy(ksi->ks_class, kp->ks_class, KSTAT_STRLEN);
539
540                 ksi->ks_instance = kp->ks_instance;
541                 ksi->ks_snaptime = kp->ks_snaptime;
542                 ksi->ks_type = kp->ks_type;
543
544                 list_create(&ksi->ks_nvlist, sizeof (ks_nvpair_t),
545                     offsetof(ks_nvpair_t, nv_next));
546
547                 SAVE_HRTIME_X(ksi, "crtime", kp->ks_crtime);
548                 SAVE_HRTIME_X(ksi, "snaptime", kp->ks_snaptime);
549                 if (g_pflg) {
550                         SAVE_STRING_X(ksi, "class", kp->ks_class);
551                 }
552
553                 /* Insert this instance into a sorted list */
554                 tmp = list_head(&instances_list);
555                 while (tmp != NULL && compare_instances(ksi, tmp) > 0)
556                         tmp = list_next(&instances_list, tmp);
557
558                 list_insert_before(&instances_list, tmp, ksi);
559
560                 /* Read the actual statistics */
561                 id = kstat_read(kc, kp, NULL);
562                 if (id == -1) {
563                         perror("kstat_read");
564                         continue;
565                 }
566
567                 switch (kp->ks_type) {
568                 case KSTAT_TYPE_RAW:
569                         save_raw(kp, ksi);
570                         break;
571                 case KSTAT_TYPE_NAMED:
572                         save_named(kp, ksi);
573                         break;
574                 case KSTAT_TYPE_INTR:
575                         save_intr(kp, ksi);
576                         break;
577                 case KSTAT_TYPE_IO:
578                         save_io(kp, ksi);
579                         break;
580                 case KSTAT_TYPE_TIMER:
581                         save_timer(kp, ksi);
582                         break;
583                 default:
584                         assert(B_FALSE); /* Invalid type */
585                         break;
586                 }
587         }
```



```
588 }
589
590 /*
591  * Print the value of a name-value pair.
592  */
593 static void
594 ks_value_print(ks_nvpair_t *nvpair)
595 {
596         switch (nvpair->data_type) {
597         case KSTAT_DATA_CHAR:
598                 (void) fprintf(stdout, "%s", nvpair->value.c);
599                 break;
600         case KSTAT_DATA_INT32:
601                 (void) fprintf(stdout, "%d", nvpair->value.i32);
602                 break;
603         case KSTAT_DATA_UINT32:
604                 (void) fprintf(stdout, "%u", nvpair->value.ui32);
605                 break;
606         case KSTAT_DATA_INT64:
607                 (void) fprintf(stdout, "%lld", nvpair->value.i64);
608                 break;
609         case KSTAT_DATA_UINT64:
610                 (void) fprintf(stdout, "%llu", nvpair->value.ui64);
611                 break;
612         case KSTAT_DATA_STRING:
613                 (void) fprintf(stdout, "%s", KSTAT_NAMED_STR_PTR(nvpair));
614                 break;
615         case KSTAT_DATA_HRTIME:
616                 if (nvpair->value.ui64 == 0)
617                         (void) fprintf(stdout, "0");
618                 else
619                         (void) fprintf(stdout, "%.9f",
620                             nvpair->value.ui64 / 1000000000.0);
621                 break;
622         default:
623                 assert(B_FALSE);
624         }
625 }
626
627 /*
628  * Print a single instance.
629  */
630 static void
631 ks_instance_print(ks_instance_t *ksi, ks_nvpair_t *nvpair)
632 {
633         if (g_headerflg) {
634                 if (!g_pflg) {
635                         (void) fprintf(stdout, DFLT_FMT,
636                             ksi->ks_module, ksi->ks_instance,
637                             ksi->ks_name, ksi->ks_class);
638                 }
639                 g_headerflg = B_FALSE;
640         }
641
642         if (g_pflg) {
643                 (void) fprintf(stdout, KS_PFMT,
644                     ksi->ks_module, ksi->ks_instance,
645                     ksi->ks_name, nvpair->name);
646                 if (!g_lflg) {
647                         (void) putchar(g_cflg ? ':': '\t');
648                         ks_value_print(nvpair);
649                 }
650         } else {
651                 (void) fprintf(stdout, KS_DFMT, nvpair->name);
652                 ks_value_print(nvpair);
653         }
```

```
655         (void) putchar('\n');
656 }

658 /*
659  * Print a single instance in JSON format.
660  */
661 static void
662 ks_instance_print_json(ks_instance_t *ksi, ks_nvpair_t *nvpair)
663 {
664         if (g_headerflg) {
665                 (void) fprintf(stdout, JSON_FMT,
666                     ksi->ks_module, ksi->ks_instance,
667                     ksi->ks_name, ksi->ks_class,
668                     ksi->ks_type);

670                 if (ksi->ks_snaptime == 0)
671                         (void) fprintf(stdout, "\t\"snaptime\": 0,\n");
672                 else
673                         (void) fprintf(stdout, "\t\"snaptime\": %.9f,\n",
674                             ksi->ks_snaptime / 1000000000.0);

676                 (void) fprintf(stdout, "\t\"data\": {\n");

678                 g_headerflg = B_FALSE;
679         }

681         (void) fprintf(stdout, KS_JFMT, nvpair->name);
682         if (nvpair->data_type == KSTAT_DATA_STRING) {
683                 (void) putchar('\"');
684                 ks_value_print(nvpair);
685                 (void) putchar('\"');
686         } else {
687                 ks_value_print(nvpair);
688         }
689         if (nvpair != list_tail(&ksi->ks_nvlist))
690                 (void) putchar(',');

692         (void) putchar('\n');
693 }

695 /*
696  * Print all instances.
697  */
698 static void
699 ks_instances_print(void)
700 {
701         ks_selector_t   *selector;
702         ks_instance_t   *ksi, *ktmp;
703         ks_nvpair_t     *nvpair, *ntmp;
704         void            (*ks_print_fn)(ks_instance_t *, ks_nvpair_t *);

706         if (g_timestamp_fmt != NODATE)
707                 print_timestamp(g_timestamp_fmt);

709         if (g_jflg) {
710                 ks_print_fn = &ks_instance_print_json;
711                 (void) putchar('[');
712         } else {
713                 ks_print_fn = &ks_instance_print;
714         }

716         /* Iterate over each selector */
717         selector = list_head(&selector_list);
718         while (selector != NULL) {
```



```
720                 /* Iterate over each instance */
721                 for (ksi = list_head(&instances_list); ksi != NULL;
722                     ksi = list_next(&instances_list, ksi)) {

724                         /* Finally iterate over each statistic */
725                         g_headerflg = B_TRUE;
726                         for (nvpair = list_head(&ksi->ks_nvlist);
727                             nvpair != NULL;
728                             nvpair = list_next(&ksi->ks_nvlist, nvpair)) {
729                                 if (gmatch(nvpair->name,
730                                     selector->ks_statistic) == 0)
731                                         continue;

733                                 g_matched = 0;
734                                 if (!g_qflg)
735                                         (*ks_print_fn)(ksi, nvpair);
736                         }

738                         if (!g_headerflg) {
739                                 if (g_jflg) {
740                                         (void) fprintf(stdout, "\t}\n}");
741                                         if (ksi != list_tail(&instances_list))
742                                                 (void) putchar(',');
743                                 } else if (!g_pflg) {
744                                         (void) putchar('\n');
745                                 }
746                         }
747                 }

749                 selector = list_next(&selector_list, selector);
750         }

752         if (g_jflg)
753                 (void) fprintf(stdout, "]\n");

755         (void) fflush(stdout);

757         /* Free the instances list */
758         ksi = list_head(&instances_list);
759         while (ksi != NULL) {
760                 nvpair = list_head(&ksi->ks_nvlist);
761                 while (nvpair != NULL) {
762                         ntmp = nvpair;
763                         nvpair = list_next(&ksi->ks_nvlist, nvpair);
764                         list_remove(&ksi->ks_nvlist, ntmp);
765                         if (ntmp->data_type == KSTAT_DATA_STRING)
766                                 free(ntmp->value.str.addr.ptr);
767                         free(ntmp);
768                 }

770                 ktmp = ksi;
771                 ksi = list_next(&instances_list, ksi);
772                 list_remove(&instances_list, ktmp);
773                 list_destroy(&ktmp->ks_nvlist);
774                 free(ktmp);
775         }
776 }

778 static void
779 save_cpu_stat(kstat_t *kp, ks_instance_t *ksi)
780 {
781         cpu_stat_t      *stat;
782         cpu_sysinfo_t   *sysinfo;
783         cpu_syswait_t   *syswait;
784         cpu_vminfo_t    *vminfo;
```

```
786         stat = (cpu_stat_t *)(kp->ks_data);
787         sysinfo = &stat->cpu_sysinfo;
788         syswait = &stat->cpu_syswait;
789         vminfo  = &stat->cpu_vminfo;

791         SAVE_UINT32_X(ksi, "idle", sysinfo->cpu[CPU_IDLE]);
792         SAVE_UINT32_X(ksi, "user", sysinfo->cpu[CPU_USER]);
793         SAVE_UINT32_X(ksi, "kernel", sysinfo->cpu[CPU_KERNEL]);
794         SAVE_UINT32_X(ksi, "wait", sysinfo->cpu[CPU_WAIT]);
795         SAVE_UINT32_X(ksi, "wait_io", sysinfo->cpu[W_IO]);
796         SAVE_UINT32_X(ksi, "wait_swap", sysinfo->cpu[W_SWAP]);
797         SAVE_UINT32_X(ksi, "wait_pio", sysinfo->cpu[W_PIO]);
798         SAVE_UINT32(ksi, sysinfo, bread);
799         SAVE_UINT32(ksi, sysinfo, bwrite);
800         SAVE_UINT32(ksi, sysinfo, lread);
801         SAVE_UINT32(ksi, sysinfo, lwrite);
802         SAVE_UINT32(ksi, sysinfo, phread);
803         SAVE_UINT32(ksi, sysinfo, phwrite);
804         SAVE_UINT32(ksi, sysinfo, pswitch);
805         SAVE_UINT32(ksi, sysinfo, trap);
806         SAVE_UINT32(ksi, sysinfo, intr);
807         SAVE_UINT32(ksi, sysinfo, syscall);
808         SAVE_UINT32(ksi, sysinfo, sysread);
809         SAVE_UINT32(ksi, sysinfo, syswrite);
810         SAVE_UINT32(ksi, sysinfo, sysfork);
811         SAVE_UINT32(ksi, sysinfo, sysvfork);
812         SAVE_UINT32(ksi, sysinfo, sysexec);
813         SAVE_UINT32(ksi, sysinfo, readch);
814         SAVE_UINT32(ksi, sysinfo, writech);
815         SAVE_UINT32(ksi, sysinfo, rcvint);
816         SAVE_UINT32(ksi, sysinfo, xmtint);
817         SAVE_UINT32(ksi, sysinfo, mdmint);
818         SAVE_UINT32(ksi, sysinfo, rawch);
819         SAVE_UINT32(ksi, sysinfo, canch);
820         SAVE_UINT32(ksi, sysinfo, outch);
821         SAVE_UINT32(ksi, sysinfo, msg);
822         SAVE_UINT32(ksi, sysinfo, sema);
823         SAVE_UINT32(ksi, sysinfo, namei);
824         SAVE_UINT32(ksi, sysinfo, ufsiget);
825         SAVE_UINT32(ksi, sysinfo, ufsdirblk);
826         SAVE_UINT32(ksi, sysinfo, ufsipage);
827         SAVE_UINT32(ksi, sysinfo, ufsinopage);
828         SAVE_UINT32(ksi, sysinfo, inodeovf);
829         SAVE_UINT32(ksi, sysinfo, fileovf);
830         SAVE_UINT32(ksi, sysinfo, procovf);
831         SAVE_UINT32(ksi, sysinfo, intrthread);
832         SAVE_UINT32(ksi, sysinfo, intrblk);
833         SAVE_UINT32(ksi, sysinfo, idlethread);
834         SAVE_UINT32(ksi, sysinfo, inv_swtch);
835         SAVE_UINT32(ksi, sysinfo, nthreads);
836         SAVE_UINT32(ksi, sysinfo, cpumigrate);
837         SAVE_UINT32(ksi, sysinfo, xcalls);
838         SAVE_UINT32(ksi, sysinfo, mutex_adenters);
839         SAVE_UINT32(ksi, sysinfo, rw_rdfails);
840         SAVE_UINT32(ksi, sysinfo, rw_wrfails);
841         SAVE_UINT32(ksi, sysinfo, modload);
842         SAVE_UINT32(ksi, sysinfo, modunload);
843         SAVE_UINT32(ksi, sysinfo, bawrite);
844 #ifdef  STATISTICS      /* see header file */
845         SAVE_UINT32(ksi, sysinfo, rw_enters);
846         SAVE_UINT32(ksi, sysinfo, win_uo_cnt);
847         SAVE_UINT32(ksi, sysinfo, win_uu_cnt);
848         SAVE_UINT32(ksi, sysinfo, win_so_cnt);
849         SAVE_UINT32(ksi, sysinfo, win_su_cnt);
850         SAVE_UINT32(ksi, sysinfo, win_suo_cnt);
851 #endif
```



```
853         SAVE_INT32(ksi, syswait, iowait);
854         SAVE_INT32(ksi, syswait, swap);
855         SAVE_INT32(ksi, syswait, physio);

857         SAVE_UINT32(ksi, vminfo, pgrec);
858         SAVE_UINT32(ksi, vminfo, pgfrec);
859         SAVE_UINT32(ksi, vminfo, pgin);
860         SAVE_UINT32(ksi, vminfo, pgpgin);
861         SAVE_UINT32(ksi, vminfo, pgout);
862         SAVE_UINT32(ksi, vminfo, pgpgout);
863         SAVE_UINT32(ksi, vminfo, swapin);
864         SAVE_UINT32(ksi, vminfo, pgswapin);
865         SAVE_UINT32(ksi, vminfo, swapout);
866         SAVE_UINT32(ksi, vminfo, pgswapout);
867         SAVE_UINT32(ksi, vminfo, zfod);
868         SAVE_UINT32(ksi, vminfo, dfree);
869         SAVE_UINT32(ksi, vminfo, scan);
870         SAVE_UINT32(ksi, vminfo, rev);
871         SAVE_UINT32(ksi, vminfo, hat_fault);
872         SAVE_UINT32(ksi, vminfo, as_fault);
873         SAVE_UINT32(ksi, vminfo, maj_fault);
874         SAVE_UINT32(ksi, vminfo, cow_fault);
875         SAVE_UINT32(ksi, vminfo, prot_fault);
876         SAVE_UINT32(ksi, vminfo, softlock);
877         SAVE_UINT32(ksi, vminfo, kernel_asflt);
878         SAVE_UINT32(ksi, vminfo, pgrrun);
879         SAVE_UINT32(ksi, vminfo, execpgin);
880         SAVE_UINT32(ksi, vminfo, execpgout);
881         SAVE_UINT32(ksi, vminfo, execfree);
882         SAVE_UINT32(ksi, vminfo, anonpgin);
883         SAVE_UINT32(ksi, vminfo, anonpgout);
884         SAVE_UINT32(ksi, vminfo, anonfree);
885         SAVE_UINT32(ksi, vminfo, fspgin);
886         SAVE_UINT32(ksi, vminfo, fspgout);
887         SAVE_UINT32(ksi, vminfo, fsfree);
888 }

890 static void
891 save_var(kstat_t *kp, ks_instance_t *ksi)
892 {
893         struct var      *var = (struct var *)(kp->ks_data);

895         assert(kp->ks_data_size == sizeof (struct var));

897         SAVE_INT32(ksi, var, v_buf);
898         SAVE_INT32(ksi, var, v_call);
899         SAVE_INT32(ksi, var, v_proc);
900         SAVE_INT32(ksi, var, v_maxupttl);
901         SAVE_INT32(ksi, var, v_nglobpris);
902         SAVE_INT32(ksi, var, v_maxsyspri);
903         SAVE_INT32(ksi, var, v_clist);
904         SAVE_INT32(ksi, var, v_maxup);
905         SAVE_INT32(ksi, var, v_hbuf);
906         SAVE_INT32(ksi, var, v_hmask);
907         SAVE_INT32(ksi, var, v_pbuf);
908         SAVE_INT32(ksi, var, v_sptmap);
909         SAVE_INT32(ksi, var, v_maxpmem);
910         SAVE_INT32(ksi, var, v_autoup);
911         SAVE_INT32(ksi, var, v_bufhwm);
912 }

914 static void
915 save_ncstats(kstat_t *kp, ks_instance_t *ksi)
916 {
917         struct ncstats  *ncstats = (struct ncstats *)(kp->ks_data);
```

```
919         assert(kp->ks_data_size == sizeof (struct ncstats));

921         SAVE_INT32(ksi, ncstats, hits);
922         SAVE_INT32(ksi, ncstats, misses);
923         SAVE_INT32(ksi, ncstats, enters);
924         SAVE_INT32(ksi, ncstats, dbl_enters);
925         SAVE_INT32(ksi, ncstats, long_enter);
926         SAVE_INT32(ksi, ncstats, long_look);
927         SAVE_INT32(ksi, ncstats, move_to_front);
928         SAVE_INT32(ksi, ncstats, purges);
929 }

931 static void
932 save_sysinfo(kstat_t *kp, ks_instance_t *ksi)
933 {
934         sysinfo_t       *sysinfo = (sysinfo_t *)(kp->ks_data);

936         assert(kp->ks_data_size == sizeof (sysinfo_t));

938         SAVE_UINT32(ksi, sysinfo, updates);
939         SAVE_UINT32(ksi, sysinfo, runque);
940         SAVE_UINT32(ksi, sysinfo, runocc);
941         SAVE_UINT32(ksi, sysinfo, swpque);
942         SAVE_UINT32(ksi, sysinfo, swpocc);
943         SAVE_UINT32(ksi, sysinfo, waiting);
944 }

946 static void
947 save_vminfo(kstat_t *kp, ks_instance_t *ksi)
948 {
949         vminfo_t        *vminfo = (vminfo_t *)(kp->ks_data);

951         assert(kp->ks_data_size == sizeof (vminfo_t));

953         SAVE_UINT64(ksi, vminfo, freemem);
954         SAVE_UINT64(ksi, vminfo, swap_resv);
955         SAVE_UINT64(ksi, vminfo, swap_alloc);
956         SAVE_UINT64(ksi, vminfo, swap_avail);
957         SAVE_UINT64(ksi, vminfo, swap_free);
958         SAVE_UINT64(ksi, vminfo, updates);
959 }

961 static void
962 save_nfs(kstat_t *kp, ks_instance_t *ksi)
963 {
964         struct mntinfo_kstat *mntinfo = (struct mntinfo_kstat *)(kp->ks_data);

966         assert(kp->ks_data_size == sizeof (struct mntinfo_kstat));

968         SAVE_STRING(ksi, mntinfo, mik_proto);
969         SAVE_UINT32(ksi, mntinfo, mik_vers);
970         SAVE_UINT32(ksi, mntinfo, mik_flags);
971         SAVE_UINT32(ksi, mntinfo, mik_secmod);
972         SAVE_UINT32(ksi, mntinfo, mik_curread);
973         SAVE_UINT32(ksi, mntinfo, mik_curwrite);
974         SAVE_INT32(ksi, mntinfo, mik_timeo);
975         SAVE_INT32(ksi, mntinfo, mik_retrans);
976         SAVE_UINT32(ksi, mntinfo, mik_acregmin);
977         SAVE_UINT32(ksi, mntinfo, mik_acregmax);
978         SAVE_UINT32(ksi, mntinfo, mik_acdirmin);
979         SAVE_UINT32(ksi, mntinfo, mik_acdirmax);
980         SAVE_UINT32_X(ksi, "lookup_srtt", mntinfo->mik_timers[0].srtt);
981         SAVE_UINT32_X(ksi, "lookup_deviate", mntinfo->mik_timers[0].deviate);
982         SAVE_UINT32_X(ksi, "lookup_rtxcur", mntinfo->mik_timers[0].rtxcur);
983         SAVE_UINT32_X(ksi, "read_srtt", mntinfo->mik_timers[1].srtt);
```



```
984         SAVE_UINT32_X(ksi, "read_deviate", mntinfo->mik_timers[1].deviate);
985         SAVE_UINT32_X(ksi, "read_rtxcur", mntinfo->mik_timers[1].rtxcur);
986         SAVE_UINT32_X(ksi, "write_srtt", mntinfo->mik_timers[2].srtt);
987         SAVE_UINT32_X(ksi, "write_deviate", mntinfo->mik_timers[2].deviate);
988         SAVE_UINT32_X(ksi, "write_rtxcur", mntinfo->mik_timers[2].rtxcur);
989         SAVE_UINT32(ksi, mntinfo, mik_noresponse);
990         SAVE_UINT32(ksi, mntinfo, mik_failover);
991         SAVE_UINT32(ksi, mntinfo, mik_remap);
992         SAVE_STRING(ksi, mntinfo, mik_curserver);
993 }

995 #ifdef __sparc
996 static void
997 save_sfmmu_global_stat(kstat_t *kp, ks_instance_t *ksi)
998 {
999         struct sfmmu_global_stat *sfmmug =
1000            (struct sfmmu_global_stat *)(kp->ks_data);

1002        assert(kp->ks_data_size == sizeof (struct sfmmu_global_stat));

1004        SAVE_INT32(ksi, sfmmug, sf_tsb_exceptions);
1005        SAVE_INT32(ksi, sfmmug, sf_tsb_raise_exception);
1006        SAVE_INT32(ksi, sfmmug, sf_pagefaults);
1007        SAVE_INT32(ksi, sfmmug, sf_uhash_searches);
1008        SAVE_INT32(ksi, sfmmug, sf_uhash_links);
1009        SAVE_INT32(ksi, sfmmug, sf_khash_searches);
1010        SAVE_INT32(ksi, sfmmug, sf_khash_links);
1011        SAVE_INT32(ksi, sfmmug, sf_swapout);
1012        SAVE_INT32(ksi, sfmmug, sf_tsb_alloc);
1013        SAVE_INT32(ksi, sfmmug, sf_tsb_allocfail);
1014        SAVE_INT32(ksi, sfmmug, sf_tsb_sectsb_create);
1015        SAVE_INT32(ksi, sfmmug, sf_scd_1sttsb_alloc);
1016        SAVE_INT32(ksi, sfmmug, sf_scd_2ndtsb_alloc);
1017        SAVE_INT32(ksi, sfmmug, sf_scd_1sttsb_allocfail);
1018        SAVE_INT32(ksi, sfmmug, sf_scd_2ndtsb_allocfail);
1019        SAVE_INT32(ksi, sfmmug, sf_tteload8k);
1020        SAVE_INT32(ksi, sfmmug, sf_tteload64k);
1021        SAVE_INT32(ksi, sfmmug, sf_tteload512k);
1022        SAVE_INT32(ksi, sfmmug, sf_tteload4m);
1023        SAVE_INT32(ksi, sfmmug, sf_tteload32m);
1024        SAVE_INT32(ksi, sfmmug, sf_tteload256m);
1025        SAVE_INT32(ksi, sfmmug, sf_tsb_load8k);
1026        SAVE_INT32(ksi, sfmmug, sf_tsb_load4m);
1027        SAVE_INT32(ksi, sfmmug, sf_hblk_hit);
1028        SAVE_INT32(ksi, sfmmug, sf_hblk8_ncreate);
1029        SAVE_INT32(ksi, sfmmug, sf_hblk8_nalloc);
1030        SAVE_INT32(ksi, sfmmug, sf_hblk1_ncreate);
1031        SAVE_INT32(ksi, sfmmug, sf_hblk1_nalloc);
1032        SAVE_INT32(ksi, sfmmug, sf_hblk_slab_cnt);
1033        SAVE_INT32(ksi, sfmmug, sf_hblk_reserve_cnt);
1034        SAVE_INT32(ksi, sfmmug, sf_hblk_recurse_cnt);
1035        SAVE_INT32(ksi, sfmmug, sf_hblk_reserve_hit);
1036        SAVE_INT32(ksi, sfmmug, sf_get_free_success);
1037        SAVE_INT32(ksi, sfmmug, sf_get_free_throttle);
1038        SAVE_INT32(ksi, sfmmug, sf_get_free_fail);
1039        SAVE_INT32(ksi, sfmmug, sf_put_free_success);
1040        SAVE_INT32(ksi, sfmmug, sf_put_free_fail);
1041        SAVE_INT32(ksi, sfmmug, sf_pgcolor_conflict);
1042        SAVE_INT32(ksi, sfmmug, sf_uncache_conflict);
1043        SAVE_INT32(ksi, sfmmug, sf_unload_conflict);
1044        SAVE_INT32(ksi, sfmmug, sf_ism_uncache);
1045        SAVE_INT32(ksi, sfmmug, sf_ism_recache);
1046        SAVE_INT32(ksi, sfmmug, sf_recache);
1047        SAVE_INT32(ksi, sfmmug, sf_steal_count);
1048        SAVE_INT32(ksi, sfmmug, sf_pagesync);
1049        SAVE_INT32(ksi, sfmmug, sf_clrwrt);
```

```
1050         SAVE_INT32(ksi, sfmmug, sf_pagesync_invalid);
1051         SAVE_INT32(ksi, sfmmug, sf_kernel_xcalls);
1052         SAVE_INT32(ksi, sfmmug, sf_user_xcalls);
1053         SAVE_INT32(ksi, sfmmug, sf_tsb_grow);
1054         SAVE_INT32(ksi, sfmmug, sf_tsb_shrink);
1055         SAVE_INT32(ksi, sfmmug, sf_tsb_resize_failures);
1056         SAVE_INT32(ksi, sfmmug, sf_tsb_reloc);
1057         SAVE_INT32(ksi, sfmmug, sf_user_vtop);
1058         SAVE_INT32(ksi, sfmmug, sf_ctx_inv);
1059         SAVE_INT32(ksi, sfmmug, sf_tlb_reprog_pgsz);
1060         SAVE_INT32(ksi, sfmmug, sf_region_remap_demap);
1061         SAVE_INT32(ksi, sfmmug, sf_create_scd);
1062         SAVE_INT32(ksi, sfmmug, sf_join_scd);
1063         SAVE_INT32(ksi, sfmmug, sf_leave_scd);
1064         SAVE_INT32(ksi, sfmmug, sf_destroy_scd);
1065 }
1066 #endif

1068 #ifdef __sparc
1069 static void
1070 save_sfmmu_tsbsize_stat(kstat_t *kp, ks_instance_t *ksi)
1071 {
1072         struct sfmmu_tsbsize_stat *sfmmut;

1074         assert(kp->ks_data_size == sizeof (struct sfmmu_tsbsize_stat));
1075         sfmmut = (struct sfmmu_tsbsize_stat *)(kp->ks_data);

1077         SAVE_INT32(ksi, sfmmut, sf_tsbsz_8k);
1078         SAVE_INT32(ksi, sfmmut, sf_tsbsz_16k);
1079         SAVE_INT32(ksi, sfmmut, sf_tsbsz_32k);
1080         SAVE_INT32(ksi, sfmmut, sf_tsbsz_64k);
1081         SAVE_INT32(ksi, sfmmut, sf_tsbsz_128k);
1082         SAVE_INT32(ksi, sfmmut, sf_tsbsz_256k);
1083         SAVE_INT32(ksi, sfmmut, sf_tsbsz_512k);
1084         SAVE_INT32(ksi, sfmmut, sf_tsbsz_1m);
1085         SAVE_INT32(ksi, sfmmut, sf_tsbsz_2m);
1086         SAVE_INT32(ksi, sfmmut, sf_tsbsz_4m);
1087 }
1088 #endif

1090 #ifdef __sparc
1091 static void
1092 save_simmstat(kstat_t *kp, ks_instance_t *ksi)
1093 {
1094         uchar_t *simmstat;
1095         char    *simm_buf;
1096         char    *list = NULL;
1097         int     i;

1099         assert(kp->ks_data_size == sizeof (uchar_t) * SIMM_COUNT);

1101         for (i = 0, simmstat = (uchar_t *)(kp->ks_data); i < SIMM_COUNT - 1;
1102             i++, simmstat++) {
1103                 if (list == NULL) {
1104                         (void) asprintf(&simm_buf, "%d,", *simmstat);
1105                 } else {
1106                         (void) asprintf(&simm_buf, "%s%d,", list, *simmstat);
1107                         free(list);
1108                 }
1109                 list = simm_buf;
1110         }

1112         (void) asprintf(&simm_buf, "%s%d", list, *simmstat);
1113         SAVE_STRING_X(ksi, "status", simm_buf);
1114         free(list);
1115         free(simm_buf);
```



```
1116 }
1117 #endif

1119 #ifdef __sparc
1120 /*
1121  * Helper function for save_temperature().
1122  */
1123 static char *
1124 short_array_to_string(short *shortp, int len)
1125 {
1126         char    *list = NULL;
1127         char    *list_buf;

1129         for (; len > 1; len--, shortp++) {
1130                 if (list == NULL) {
1131                         (void) asprintf(&list_buf, "%d,", *shortp);
1132                 } else {
1133                         (void) asprintf(&list_buf, "%s%d,", list, *shortp);
1134                         free(list);
1135                 }
1136                 list = list_buf;
1137         }

1139         (void) asprintf(&list_buf, "%s%s", list, *shortp);
1140         free(list);
1141         return (list_buf);
1142 }

1144 static void
1145 save_temperature(kstat_t *kp, ks_instance_t *ksi)
1146 {
1147         struct temp_stats *temps = (struct temp_stats *)(kp->ks_data);
1148         char    *buf;
1149         int     n = 1;

1151         assert(kp->ks_data_size == sizeof (struct temp_stats));

1153         SAVE_UINT32(ksi, temps, index);

1155         buf = short_array_to_string(temps->l1, L1_SZ);
1156         SAVE_STRING_X(ksi, "l1", buf);
1157         free(buf);

1159         buf = short_array_to_string(temps->l2, L2_SZ);
1160         SAVE_STRING_X(ksi, "l2", buf);
1161         free(buf);

1163         buf = short_array_to_string(temps->l3, L3_SZ);
1164         SAVE_STRING_X(ksi, "l3", buf);
1165         free(buf);

1167         buf = short_array_to_string(temps->l4, L4_SZ);
1168         SAVE_STRING_X(ksi, "l4", buf);
1169         free(buf);

1171         buf = short_array_to_string(temps->l5, L5_SZ);
1172         SAVE_STRING_X(ksi, "l5", buf);
1173         free(buf);

1175         SAVE_INT32(ksi, temps, max);
1176         SAVE_INT32(ksi, temps, min);
1177         SAVE_INT32(ksi, temps, state);
1178         SAVE_INT32(ksi, temps, temp_cnt);
1179         SAVE_INT32(ksi, temps, shutdown_cnt);
1180         SAVE_INT32(ksi, temps, version);
1181         SAVE_INT32(ksi, temps, trend);
```

```
1182         SAVE_INT32(ksi, temps, override);
1183 }
1184 #endif

1186 #ifdef __sparc
1187 static void
1188 save_temp_over(kstat_t *kp, ks_instance_t *ksi)
1189 {
1190         short   *sh = (short *)(kp->ks_data);
1191         char    *value;

1193         assert(kp->ks_data_size == sizeof (short));

1195         (void) asprintf(&value, "%hu", *sh);
1196         SAVE_STRING_X(ksi, "override", value);
1197         free(value);
1198 }
1199 #endif

1201 #ifdef __sparc
1202 static void
1203 save_ps_shadow(kstat_t *kp, ks_instance_t *ksi)
1204 {
1205         uchar_t *uchar = (uchar_t *)(kp->ks_data);

1207         assert(kp->ks_data_size == SYS_PS_COUNT);

1209         SAVE_CHAR_X(ksi, "core_0", *uchar++);
1210         SAVE_CHAR_X(ksi, "core_1", *uchar++);
1211         SAVE_CHAR_X(ksi, "core_2", *uchar++);
1212         SAVE_CHAR_X(ksi, "core_3", *uchar++);
1213         SAVE_CHAR_X(ksi, "core_4", *uchar++);
1214         SAVE_CHAR_X(ksi, "core_5", *uchar++);
1215         SAVE_CHAR_X(ksi, "core_6", *uchar++);
1216         SAVE_CHAR_X(ksi, "core_7", *uchar++);
1217         SAVE_CHAR_X(ksi, "pps_0", *uchar++);
1218         SAVE_CHAR_X(ksi, "clk_33", *uchar++);
1219         SAVE_CHAR_X(ksi, "clk_50", *uchar++);
1220         SAVE_CHAR_X(ksi, "v5_p", *uchar++);
1221         SAVE_CHAR_X(ksi, "v12_p", *uchar++);
1222         SAVE_CHAR_X(ksi, "v5_aux", *uchar++);
1223         SAVE_CHAR_X(ksi, "v5_p_pch", *uchar++);
1224         SAVE_CHAR_X(ksi, "v12_p_pch", *uchar++);
1225         SAVE_CHAR_X(ksi, "v3_pch", *uchar++);
1226         SAVE_CHAR_X(ksi, "v5_pch", *uchar++);
1227         SAVE_CHAR_X(ksi, "p_fan", *uchar++);
1228 }
1229 #endif

1231 #ifdef __sparc
1232 static void
1233 save_fault_list(kstat_t *kp, ks_instance_t *ksi)
1234 {
1235         struct ft_list *fault;
1236         char    name[KSTAT_STRLEN + 7];
1237         int     i;

1239         for (i = 1, fault = (struct ft_list *)(kp->ks_data);
1240             i <= 999999 && i <= kp->ks_data_size / sizeof (struct ft_list);
1241             i++, fault++) {
1242                 (void) snprintf(name, sizeof (name), "unit_%d", i);
1243                 SAVE_INT32_X(ksi, name, fault->unit);
1244                 (void) snprintf(name, sizeof (name), "type_%d", i);
1245                 SAVE_INT32_X(ksi, name, fault->type);
1246                 (void) snprintf(name, sizeof (name), "fclass_%d", i);
1247                 SAVE_INT32_X(ksi, name, fault->fclass);
```



```
1248                 (void) snprintf(name, sizeof (name), "create_time_%d", i);
1249                 SAVE_HRTIME_X(ksi, name, fault->create_time);
1250                 (void) snprintf(name, sizeof (name), "msg_%d", i);
1251                 SAVE_STRING_X(ksi, name, faultp->msg);
1252         }
1253 }
1254 #endif

1256 static void
1257 save_named(kstat_t *kp, ks_instance_t *ksi)
1258 {
1259         kstat_named_t *knp;
1260         int     n;

1262         for (n = kp->ks_ndata, knp = KSTAT_NAMED_PTR(kp); n > 0; n--, knp++) {
1263                 switch (knp->data_type) {
1264                 case KSTAT_DATA_CHAR:
1265                         nvpair_insert(ksi, knp->name,
1266                             (ks_value_t *) &knp->value, KSTAT_DATA_CHAR);
1267                         break;
1268                 case KSTAT_DATA_INT32:
1269                         nvpair_insert(ksi, knp->name,
1270                             (ks_value_t *) &knp->value, KSTAT_DATA_INT32);
1271                         break;
1272                 case KSTAT_DATA_UINT32:
1273                         nvpair_insert(ksi, knp->name,
1274                             (ks_value_t *) &knp->value, KSTAT_DATA_UINT32);
1275                         break;
1276                 case KSTAT_DATA_INT64:
1277                         nvpair_insert(ksi, knp->name,
1278                             (ks_value_t *) &knp->value, KSTAT_DATA_INT64);
1279                         break;
1280                 case KSTAT_DATA_UINT64:
1281                         nvpair_insert(ksi, knp->name,
1282                             (ks_value_t *) &knp->value, KSTAT_DATA_UINT64);
1283                         break;
1284                 case KSTAT_DATA_STRING:
1285                         SAVE_STRING_X(ksi, knp->name, KSTAT_NAMED_STR_PTR(knp));
1286                         break;
1287                 default:
1288                         assert(B_FALSE); /* Invalid data type */
1289                         break;
1290                 }
1291         }
1292 }

1294 static void
1295 save_intr(kstat_t *kp, ks_instance_t *ksi)
1296 {
1297         kstat_intr_t *intr = KSTAT_INTR_PTR(kp);
1298         char    *intr_names[] = {"hard", "soft", "watchdog", "spurious",
1299             "multiple_service"};
1300         int     n;

1302         for (n = 0; n < KSTAT_NUM_INTRS; n++)
1303                 SAVE_UINT32_X(ksi, intr_names[n], intr->intrs[n]);
1304 }

1306 static void
1307 save_io(kstat_t *kp, ks_instance_t *ksi)
1308 {
1309         kstat_io_t      *ksio = KSTAT_IO_PTR(kp);

1311         SAVE_UINT64(ksi, ksio, nread);
1312         SAVE_UINT64(ksi, ksio, nwritten);
1313         SAVE_UINT32(ksi, ksio, reads);
```

```
1314         SAVE_UINT32(ksi, ksio, writes);
1315         SAVE_HRTIME(ksi, ksio, wtime);
1316         SAVE_HRTIME(ksi, ksio, wlentime);
1317         SAVE_HRTIME(ksi, ksio, wlastupdate);
1318         SAVE_HRTIME(ksi, ksio, rtime);
1319         SAVE_HRTIME(ksi, ksio, rlentime);
1320         SAVE_HRTIME(ksi, ksio, rlastupdate);
1321         SAVE_UINT32(ksi, ksio, wcnt);
1322         SAVE_UINT32(ksi, ksio, rcnt);
1323 }

1325 static void
1326 save_timer(kstat_t *kp, ks_instance_t *ksi)
1327 {
1328         kstat_timer_t   *ktimer = KSTAT_TIMER_PTR(kp);

1330         SAVE_STRING(ksi, ktimer, name);
1331         SAVE_UINT64(ksi, ktimer, num_events);
1332         SAVE_HRTIME(ksi, ktimer, elapsed_time);
1333         SAVE_HRTIME(ksi, ktimer, min_time);
1334         SAVE_HRTIME(ksi, ktimer, max_time);
1335         SAVE_HRTIME(ksi, ktimer, start_time);
1336         SAVE_HRTIME(ksi, ktimer, stop_time);
1337 }
1338 #endif /* ! codereview */
```

```
**********************************************************
    6700 Thu Nov 22 14:24:27 2012
new/usr/src/cmd/stat/kstat/kstat.h
749 "/usr/bin/kstat" should be rewritten in C
Reviewed by: Garrett D'Amore <garrett@damore.org>
Reviewed by: Brendan Gregg <brendan.gregg@joyent.com>
**********************************************************
```

```
/*
 * 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.
 * Copyright 2012 David Hoeppner.  All rights reserved.
 */

#ifndef _STAT_KSTAT_H
#define _STAT_KSTAT_H

/*
 * Structures needed by the kstat reader functions
 */
#include <sys/var.h>
#include <sys/utsname.h>
#include <sys/sysinfo.h>
#include <sys/flock.h>
#include <sys/dnlc.h>
#include <nfs/nfs.h>
#include <nfs/nfs_clnt.h>

#ifdef __sparc
#include <vm/hat_sfmmu.h>
#include <sys/simmstat.h>
#include <sys/sysctrl.h>
#include <sys/fhc.h>
#endif

#define KSTAT_DATA_HRTIME       (KSTAT_DATA_STRING + 1)

typedef union ks_value {
        char            c[16];
        int32_t         i32;
        uint32_t        ui32;
        struct {
                union {
                        char    *ptr;
                        char    __pad[8];
                } addr;
                uint32_t        len;
        } str;
```



```
        int64_t         i64;
        uint64_t        ui64;
} ks_value_t;

#define SAVE_HRTIME(I, S, N)                                    \
{                                                               \
        ks_value_t v;                                           \
        v.ui64 = S->N;                                          \
        nvpair_insert(I, #N, &v, KSTAT_DATA_UINT64);    \
}

#define SAVE_INT32(I, S, N)                                     \
{                                                               \
        ks_value_t v;                                           \
        v.i32 = S->N;                                           \
        nvpair_insert(I, #N, &v, KSTAT_DATA_INT32);     \
}

#define SAVE_UINT32(I, S, N)                                    \
{                                                               \
        ks_value_t v;                                           \
        v.ui32 = S->N;                                          \
        nvpair_insert(I, #N, &v, KSTAT_DATA_UINT32);    \
}

#define SAVE_INT64(I, S, N)                                     \
{                                                               \
        ks_value_t v;                                           \
        v.i64 = S->N;                                           \
        nvpair_insert(I, #N, &v, KSTAT_DATA_INT64);     \
}

#define SAVE_UINT64(I, S, N)                                    \
{                                                               \
        ks_value_t v;                                           \
        v.ui64 = S->N;                                          \
        nvpair_insert(I, #N, &v, KSTAT_DATA_UINT64);    \
}

/*
 * We dont want const "strings" because we free
 * the instances later
 */
#define SAVE_STRING(I, S, N)                                    \
{                                                               \
        ks_value_t v;                                           \
        v.str.addr.ptr = safe_strdup(S->N);             \
        v.str.len = strlen(S->N);                       \
        nvpair_insert(I, #N, &v, KSTAT_DATA_STRING);    \
}

#define SAVE_HRTIME_X(I, N, V)                                  \
{                                                               \
        ks_value_t v;                                           \
        v.ui64 = V;                                             \
        nvpair_insert(I, N, &v, KSTAT_DATA_HRTIME);     \
}

#define SAVE_INT32_X(I, N, V)                                   \
{                                                               \
        ks_value_t v;                                           \
        v.i32 = V;                                              \
        nvpair_insert(I, N, &v, KSTAT_DATA_INT32);      \
}
```

```
126 #define SAVE_UINT32_X(I, N, V)                          \
127 {                                                       \
128         ks_value_t v;                                   \
129         v.ui32 = V;                                     \
130         nvpair_insert(I, N, &v, KSTAT_DATA_UINT32);     \
131 }

133 #define SAVE_UINT64_X(I, N, V)                          \
134 {                                                       \
135         ks_value_t v;                                   \
136         v.ui64 = V;                                     \
137         nvpair_insert(I, N, &v, KSTAT_DATA_UINT64);     \
138 }

140 #define SAVE_STRING_X(I, N, V)                          \
141 {                                                       \
142         ks_value_t v;                                   \
143         v.str.addr.ptr = safe_strdup(V);                \
144         v.str.len = strlen(V);                          \
145         nvpair_insert(I, N, &v, KSTAT_DATA_STRING);     \
146 }

148 #define SAVE_CHAR_X(I, N, V)                            \
149 {                                                       \
150         ks_value_t v;                                   \
151         asprintf(&v.str.addr.ptr, "%c", V);             \
152         v.str.len = 1;                                  \
153         nvpair_insert(I, N, &v, KSTAT_DATA_STRING);     \
154 }

156 #define DFLT_FMT                                        \
157         "module: %-30.30s  instance: %-6d\n"            \
158         "name:   %-30.30s  class:    %-.30s\n"

160 #define JSON_FMT                                        \
161         "{\n\t\"module\": \"%s\",\n"                    \
162         "\t\"instance\": %d,\n"                         \
163         "\t\"name\": \"%s\",\n"                         \
164         "\t\"class\": \"%s\",\n"                        \
165         "\t\"type\": %d,\n"

167 #define KS_DFMT "\t%-30s  "
168 #define KS_JFMT "\t\t\"%s\": "
169 #define KS_PFMT "%s:%d:%s:%s"

171 typedef struct ks_instance {
172         list_node_t     ks_next;
173         char            ks_name[KSTAT_STRLEN];
174         char            ks_module[KSTAT_STRLEN];
175         char            ks_class[KSTAT_STRLEN];
176         int             ks_instance;
177         uchar_t         ks_type;
178         hrtime_t        ks_snaptime;
179         list_t          ks_nvlist;
180 } ks_instance_t;

182 typedef struct ks_nvpair {
183         list_node_t     nv_next;
184         char            name[KSTAT_STRLEN];
185         uchar_t         data_type;
186         ks_value_t      value;
187 } ks_nvpair_t;

189 typedef struct ks_selector {
190         list_node_t ks_next;
191         char    *ks_module;
```



```
192         char    *ks_instance;
193         char    *ks_name;
194         char    *ks_statistic;
195 } ks_selector_t;

197 static void     usage(void);
198 static int      compare_instances(ks_instance_t *, ks_instance_t *);
199 static void     nvpair_insert(ks_instance_t *, char *, ks_value_t *, uchar_t);
200 static ks_selector_t    *new_selector(void);
201 static void     ks_instances_read(kstat_ctl_t *);
202 static void     ks_value_print(ks_nvpair_t *);
203 static void     ks_instance_print(ks_instance_t *, ks_nvpair_t *);
204 static void     ks_instances_print(void);

206 /* Raw kstat readers */
207 static void     save_cpu_stat(kstat_t *, ks_instance_t *);
208 static void     save_var(kstat_t *, ks_instance_t *);
209 static void     save_ncstats(kstat_t *, ks_instance_t *);
210 static void     save_sysinfo(kstat_t *, ks_instance_t *);
211 static void     save_vminfo(kstat_t *, ks_instance_t *);
212 static void     save_nfs(kstat_t *, ks_instance_t *);
213 #ifdef __sparc
214 static void     save_sfmmu_global_stat(kstat_t *, ks_instance_t *);
215 static void     save_sfmmu_tsbsize_stat(kstat_t *, ks_instance_t *);
216 static void     save_simmstat(kstat_t *, ks_instance_t *);
217 /* Helper function for save_temperature() */
218 static char     *short_array_to_string(short *, int);
219 static void     save_temperature(kstat_t *, ks_instance_t *);
220 static void     save_temp_over(kstat_t *, ks_instance_t *);
221 static void     save_ps_shadow(kstat_t *, ks_instance_t *);
222 static void     save_fault_list(kstat_t *, ks_instance_t *);
223 #endif

225 /* Named kstat readers */
226 static void     save_named(kstat_t *, ks_instance_t *);
227 static void     save_intr(kstat_t *, ks_instance_t *);
228 static void     save_io(kstat_t *, ks_instance_t *);
229 static void     save_timer(kstat_t *, ks_instance_t *);

231 /* Typedef for raw kstat reader functions */
232 typedef void    (*kstat_raw_reader_t)(kstat_t *, ks_instance_t *);

234 static struct {
235         kstat_raw_reader_t fn;
236         char *name;
237 } ks_raw_lookup[] = {
238         /* Function name                kstat name              */
239         {save_cpu_stat,                 "cpu_stat:cpu_stat"},
240         {save_var,                      "unix:var"},
241         {save_ncstats,                  "unix:ncstats"},
242         {save_sysinfo,                  "unix:sysinfo"},
243         {save_vminfo,                   "unix:vminfo"},
244         {save_nfs,                      "nfs:mntinfo"},
245 #ifdef __sparc
246         {save_sfmmu_global_stat,        "unix:sfmmu_global_stat"},
247         {save_sfmmu_tsbsize_stat,       "unix:sfmmu_tsbsize_stat"},
248         {save_simmstat,                 "unix:simm-status"},
249         {save_temperature,              "unix:temperature"},
250         {save_temp_over,                "unix:temperature override"},
251         {save_ps_shadow,                "unix:ps_shadow"},
252         {save_fault_list,               "unix:fault_list"},
253 #endif
254         {NULL, NULL},
255 };

257 static kstat_raw_reader_t       lookup_raw_kstat_fn(char *, char *);
```

```
259 #endif /* _STAT_KSTAT_H */
260 #endif /* ! codereview */
```

```
**********************************************************
    8971 Thu Nov 22 14:24:27 2012
new/usr/src/man/man1m/kstat.1m
749 "/usr/bin/kstat" should be rewritten in C
Reviewed by: Garrett D'Amore <garrett@damore.org>
Reviewed by: Brendan Gregg <brendan.gregg@joyent.com>
**********************************************************
'\" te
.\" Copyright (c) 2000, Sun Microsystems, Inc. All Rights Reserved
.\" The contents of this file are subject to the terms of the Common Development
.\"  See the License for the specific language governing permissions and limitat
.\" the fields enclosed by brackets "[]" replaced with your own identifying info
.TH KSTAT 1M "Nov 22, 2012"
.TH KSTAT 1M "Mar 23, 2009"
.SH NAME
kstat \- display kernel statistics
.SH SYNOPSIS
.LP
.nf
\fBkstat\fR [\fB-Cjlpq\fR] [\fB-T\fR u | d ] [\fB-c\fR \fIclass\fR] [\fB-m\fR \f
\fBkstat\fR [\fB-lpq\fR] [\fB-T\fR u | d ] [\fB-c\fR \fIclass\fR] [\fB-m\fR \fIm
     [\fB-i\fR \fIinstance\fR] [\fB-n\fR \fIname\fR] [\fB-s\fR \fIstatistic\fR]
     [interval [count]]
.fi

.LP
.nf
\fBkstat\fR [\fB-Cjlpq\fR] [\fB-T\fR u | d ] [\fB-c\fR \fIclass\fR]
\fBkstat\fR [\fB-lpq\fR] [\fB-T\fR u | d ] [\fB-c\fR \fIclass\fR]
     [\fImodule\fR:\fIinstance\fR:\fIname\fR:\fIstatistic\fR]...
     [interval [count]]
.fi

.SH DESCRIPTION
.sp
.LP
The \fBkstat\fR utility examines the available kernel statistics, or kstats, on
the system and reports those statistics which match the criteria specified on
the command line. Each matching statistic is printed with its module, instance,
and name fields, as well as its actual value.
.sp
.LP
Kernel statistics may be published by various kernel subsystems, such as
drivers or loadable modules; each kstat has a module field that denotes its
publisher. Since each module might have countable entities (such as multiple
disks associated with the \fBsd\fR(7D) driver) for which it wishes to report
statistics, the kstat also has an instance field to index the statistics for
each entity; kstat instances are numbered starting from zero. Finally, the
kstat is given a name unique within its module.
.sp
.LP
Each kstat may be a special kstat type, an array of name-value pairs, or raw
data. In the name-value case, each reported value is given a label, which we
refer to as the statistic. Known raw and special kstats are given statistic
labels for each of their values by \fBkstat\fR; thus, all published values can
be referenced as \fImodule\fR:\fIinstance\fR:\fIname\fR:\fIstatistic\fR.
.sp
.LP
When invoked without any module operands or options, kstat will match all
defined statistics on the system. Example invocations are provided below. All
times are displayed as fractional seconds since system boot.
.SH OPTIONS
.sp
.LP
The tests specified by the following options are logically ANDed, and all
matching kstats will be selected. A regular expression containing shell
```



```
metacharacters must be protected from the shell by enclosing it with the
appropriate quotes.
.sp
.LP
The argument for the \fB-c\fR, \fB-i\fR, \fB-m\fR, \fB-n\fR, and \fB-s\fR
options may be specified as a shell glob pattern.
.sp
.ne 2
.na
\fB\fB-C\fR\fR
.ad
.RS 16n
Displays output in parseable format with a colon as separator.
.RE

options may be specified as a shell glob pattern, or a Perl regular expression
enclosed in '/' characters.
.sp
.ne 2
.na
\fB\fB-c\fR \fIclass\fR\fR
.ad
.RS 16n
Displays only kstats that match the specified class. \fIclass\fR is a
kernel-defined string which classifies the "type" of the kstat.
.RE

.sp
.ne 2
.na
\fB\fB-i\fR \fIinstance\fR\fR
.ad
.RS 16n
Displays only kstats that match the specified instance.
.RE

.sp
.ne 2
.na
\fB\fB-j\fR\fR
.ad
.RS 16n
Displays output in JSON format.
.RE

.sp
.ne 2
.na
#endif /* ! codereview */
\fB\fB-l\fR\fR
.ad
.RS 16n
Lists matching kstat names without displaying values.
.RE

.sp
.ne 2
.na
\fB\fB-m\fR \fImodule\fR\fR
.ad
.RS 16n
Displays only kstats that match the specified module.
.RE

.sp
.ne 2
```

```
.na
\fB\fB-n\fR \fIname\fR\fR
.ad
.RS 16n
Displays only kstats that match the specified name.
.RE

.sp
.ne 2
.na
\fB\fB-p\fR\fR
.ad
.RS 16n
Displays output in parseable format. All example output in this document is
given in this format. If this option is not specified, \fBkstat\fR produces
output in a human-readable, table format.
.RE

.sp
.ne 2
.na
\fB\fB-q\fR\fR
.ad
.RS 16n
Displays no output, but return appropriate exit status for matches against
given criteria.
.RE

.sp
.ne 2
.na
\fB\fB-s\fR \fIstatistic\fR\fR
.ad
.RS 16n
Displays only kstats that match the specified statistic.
.RE

.sp
.ne 2
.na
\fB\fB-T\fR d | u\fR
.ad
.RS 16n
Displays a time stamp before each statistics block, either in \fBdate\fR(1)
format (\fBd\fR) or as an alphanumeric representation of the value returned by
\fBtime\fR(2) (\fBu\fR).
.RE

.SH OPERANDS
.sp
.LP
The following operands are supported:
.sp
.ne 2
.na
\fB\fImodule\fR:\fIinstance\fR:\fIname\fR:\fIstatistic\fR\fR
.ad
.sp .6
.RS 4n
Alternate method of specifying module, instance, name, and statistic as
described above. Each of the module, instance, name, or statistic specifiers
may be a shell glob pattern.
It is possible to use both specifier types within a single operand.
 may be a shell glob pattern or a Perl regular expression enclosed by '/'
 characters. It is possible to use both specifier types within a single operand.
Leaving a specifier empty is equivalent to using the '*' glob pattern for that
```



```
specifier.
.RE

.sp
.ne 2
.na
\fB\fIinterval\fR\fR
.ad
.sp .6
.RS 4n
The number of seconds between reports.
.RE

.sp
.ne 2
.na
\fB\fIcount\fR\fR
.ad
.sp .6
.RS 4n
The number of reports to be printed.
.RE

.SH EXAMPLES
.sp
.LP
In the following examples, all the command lines in a block produce the same
output, as shown immediately below. The exact statistics and values will of
course vary from machine to machine.
.LP
\fBExample 1 \fRUsing the \fBkstat\fR Command
.sp
.in +2
.nf
example$ \fBkstat -p -m unix -i 0 -n system_misc -s 'avenrun*'\fR
example$ \fBkstat -p -s 'avenrun*'\fR
example$ \fBkstat -p 'unix:0:system_misc:avenrun*'\fR
example$ \fBkstat -p ':::avenrun*'\fR
example$ \fBkstat -p ':::avenrun*min$'\fR
example$ \fBkstat -p ':::/^avenrun_\ed+min$/'\fR

unix:0:system_misc:avenrun_15min        3
unix:0:system_misc:avenrun_1min 4
unix:0:system_misc:avenrun_5min 2
.fi
.in -2
.sp

.LP
\fBExample 2 \fRUsing the \fBkstat\fR Command
.sp
.in +2
.nf
example$ \fBkstat -p -m cpu_stat -s 'intr*'\fR
example$ \fBkstat -p 'cpu_stat:::intr*'\fR
example$ \fBkstat -p cpu_stat:::/^intr/\fR

cpu_stat:0:cpu_stat0:intr       29682330
cpu_stat:0:cpu_stat0:intrblk    87
cpu_stat:0:cpu_stat0:intrthread 15054222
cpu_stat:1:cpu_stat1:intr       426073
cpu_stat:1:cpu_stat1:intrblk    51
cpu_stat:1:cpu_stat1:intrthread 289668
cpu_stat:2:cpu_stat2:intr       134160
cpu_stat:2:cpu_stat2:intrblk    0
cpu_stat:2:cpu_stat2:intrthread 131
```

```
249 cpu_stat:3:cpu_stat3:intr       196566
250 cpu_stat:3:cpu_stat3:intrblk    30
251 cpu_stat:3:cpu_stat3:intrthread 59626
252 .fi
253 .in -2
254 .sp

256 .LP
257 \fBExample 3 \fRUsing the \fBkstat\fR Command
258 .sp
259 .in +2
260 .nf
261 example$ \fBkstat -p :::state ':::avenrun*'\fR
166 example$ \fBkstat -p :::state :::/^avenrun/\fR

263 cpu_info:0:cpu_info0:state      on-line
264 cpu_info:1:cpu_info1:state      on-line
265 cpu_info:2:cpu_info2:state      on-line
266 cpu_info:3:cpu_info3:state      on-line
267 unix:0:system_misc:avenrun_15min        4
268 unix:0:system_misc:avenrun_1min 10
269 unix:0:system_misc:avenrun_5min 3
270 .fi
271 .in -2
272 .sp

274 .LP
275 \fBExample 4 \fRUsing the \fBkstat\fR Command
276 .sp
277 .in +2
278 .nf
279 example$ \fBkstat -p 'unix:0:system_misc:avenrun*' 1 3\fR
280 unix:0:system_misc:avenrun_15min        15
281 unix:0:system_misc:avenrun_1min 11
282 unix:0:system_misc:avenrun_5min 21

284 unix:0:system_misc:avenrun_15min        15
285 unix:0:system_misc:avenrun_1min 11
286 unix:0:system_misc:avenrun_5min 21

288 unix:0:system_misc:avenrun_15min        15
289 unix:0:system_misc:avenrun_1min 11
290 unix:0:system_misc:avenrun_5min 21
291 .fi
292 .in -2
293 .sp

295 .LP
296 \fBExample 5 \fRUsing the \fBkstat\fR Command
297 .sp
298 .in +2
299 .nf
300 example$ \fBkstat -p -T d 'unix:0:system_misc:avenrun*' 5 2\fR
301 Thu Jul 22 19:39:50 1999
302 unix:0:system_misc:avenrun_15min        12
303 unix:0:system_misc:avenrun_1min 0
304 unix:0:system_misc:avenrun_5min 11

306 Thu Jul 22 19:39:55 1999
307 unix:0:system_misc:avenrun_15min        12
308 unix:0:system_misc:avenrun_1min 0
309 unix:0:system_misc:avenrun_5min 11
310 .fi
311 .in -2
312 .sp
```



```
314 .LP
315 \fBExample 6 \fRUsing the \fBkstat\fR Command
316 .sp
317 .in +2
318 .nf
319 example$ \fBkstat -p -T u 'unix:0:system_misc:avenrun*'\fR
320 932668656
321 unix:0:system_misc:avenrun_15min        14
322 unix:0:system_misc:avenrun_1min 5
323 unix:0:system_misc:avenrun_5min 18
324 .fi
325 .in -2
326 .sp

328 .SH EXIT STATUS
329 .sp
330 .LP
331 The following exit values are returned:
332 .sp
333 .ne 2
334 .na
335 \fB\fB0\fR\fR
336 .ad
337 .RS 5n
338 One or more statistics were matched.
339 .RE

341 .sp
342 .ne 2
343 .na
344 \fB\fB1\fR\fR
345 .ad
346 .RS 5n
347 No statistics were matched.
348 .RE

350 .sp
351 .ne 2
352 .na
353 \fB\fB2\fR\fR
354 .ad
355 .RS 5n
356 Invalid command line options were specified.
357 .RE

359 .sp
360 .ne 2
361 .na
362 \fB\fB3\fR\fR
363 .ad
364 .RS 5n
365 A fatal error occurred.
366 .RE

368 .SH FILES
369 .sp
370 .ne 2
371 .na
372 \fB\fB/dev/kstat\fR\fR
373 .ad
374 .RS 14n
375 kernel statistics driver
376 .RE

378 .SH SEE ALSO
379 .sp
```

```
 380 .LP
 381 \fBdate\fR(1), \fBsh\fR(1), \fBtime\fR(2), \fBgmatch\fR(3GEN),
 382 \fBkstat\fR(3KSTAT), \fBattributes\fR(5), \fBkstat\fR(7D), \fBsd\fR(7D),
 383 \fBkstat\fR(9S)
 384 .SH NOTES
 385 .sp
 386 .LP
 387 If the pattern argument contains glob metacharacters which are also
 292 If the pattern argument contains glob or Perl RE metacharacters which are also
 388 shell metacharacters, it will be necessary to enclose the pattern with
 389 appropriate shell quotes.
```