```
**********************************************************
    1267 Wed Nov 28 23:08:57 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>
kstat
**********************************************************
```

```
   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 Wed Nov 28 23:08:57 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>
kstat
**********************************************************
```

```
#
# 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 */
```

```
**********************************************************
   34125 Wed Nov 28 23:08:58 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>
kstat
**********************************************************
```

```
/*
 * 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 <signal.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";     /* Name of this command */
int     caught_cont = 0;        /* Have caught a SIGCONT */

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.
```

```
125          */
126         while ((c = getopt(argc, argv, "h?CqjlpT:m:i:n:s:c:")) != EOF)
127                 switch (c) {
128                 case 'h':
129                 case '?':
130                         usage();
131                         exit(0);
132                         break;
133                 case 'C':
134                         g_pflg = g_cflg = B_TRUE;
135                         break;
136                 case 'q':
137                         g_qflg = B_TRUE;
138                         break;
139                 case 'j':
140                         g_jflg = B_TRUE;
141                         break;
142                 case 'l':
143                         g_pflg = g_lflg = B_TRUE;
144                         break;
145                 case 'p':
146                         g_pflg = B_TRUE;
147                         break;
148                 case 'T':
149                         switch (*optarg) {
150                         case 'd':
151                                 g_timestamp_fmt = DDATE;
152                                 break;
153                         case 'u':
154                                 g_timestamp_fmt = UDATE;
155                                 break;
156                         default:
157                                 errflg = B_TRUE;
158                         }
159                         break;
160                 case 'm':
161                         nselflg = B_TRUE;
162                         nselector->ks_module =
163                             (char *)safe_strdup(optarg);
164                         break;
165                 case 'i':
166                         nselflg = B_TRUE;
167                         nselector->ks_instance =
168                             (char *)safe_strdup(optarg);
169                         break;
170                 case 'n':
171                         nselflg = B_TRUE;
172                         nselector->ks_name =
173                             (char *)safe_strdup(optarg);
174                         break;
175                 case 's':
176                         nselflg = B_TRUE;
177                         nselector->ks_statistic =
178                             (char *)safe_strdup(optarg);
179                         break;
180                 case 'c':
181                         g_ks_class =
182                             (char *)safe_strdup(optarg);
183                         break;
184                 default:
185                         errflg = B_TRUE;
186                         break;
187                 }

189         if (g_qflg && (g_jflg || g_pflg)) {
190                 (void) fprintf(stderr, gettext(
```



```
191                     "-q and -lpj are mutually exclusive\n"));
192                 errflg = B_TRUE;
193         }

195         if (errflg) {
196                 usage();
197                 exit(2);
198         }

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

203         /*
204          * Consume the rest of the command line. Parsing the
205          * unnamed command line arguments.
206          */
207         while (argc--) {
208                 errno = 0;
209                 tmp = strtoul(*argv, &q, 10);
210                 if (tmp == ULONG_MAX && errno == ERANGE) {
211                         if (n == 0) {
212                                 (void) fprintf(stderr, gettext(
213                                     "Interval is too large\n"));
214                         } else if (n == 1) {
215                                 (void) fprintf(stderr, gettext(
216                                     "Count is too large\n"));
217                         }
218                         usage();
219                         exit(2);
220                 }

222                 if (errno != 0 || *q != '\0') {
223                         m = 0;
224                         uselector = new_selector();
225                         while ((q = (char *)strsep(argv, ":")) != NULL) {
226                                 m++;
227                                 if (m > 4) {
228                                         free(uselector);
229                                         usage();
230                                         exit(2);
231                                 }

233                                 if (*q != '\0') {
234                                         switch (m) {
235                                         case 1:
236                                                 uselector->ks_module =
237                                                     (char *)safe_strdup(q);
238                                                 break;
239                                         case 2:
240                                                 uselector->ks_instance =
241                                                     (char *)safe_strdup(q);
242                                                 break;
243                                         case 3:
244                                                 uselector->ks_name =
245                                                     (char *)safe_strdup(q);
246                                                 break;
247                                         case 4:
248                                                 uselector->ks_statistic =
249                                                     (char *)safe_strdup(q);
250                                                 break;
251                                         default:
252                                                 assert(B_FALSE);
253                                         }
254                                 }
255                         }
```

```
257                         if (m < 4) {
258                                 free(uselector);
259                                 usage();
260                                 exit(2);
261                         }

263                         uselflg = B_TRUE;
264                         list_insert_tail(&selector_list, uselector);
265                 } else {
266                         if (tmp < 1) {
267                                 if (n == 0) {
268                                         (void) fprintf(stderr, gettext(
269                                             "Interval must be an "
270                                             "integer >= 1"));
271                                 } else if (n == 1) {
272                                         (void) fprintf(stderr, gettext(
273                                             "Count must be an integer >= 1"));
274                                 }
275                                 usage();
276                                 exit(2);
277                         } else {
278                                 if (n == 0) {
279                                         interval = tmp;
280                                         count = -1;
281                                 } else if (n == 1) {
282                                         count = tmp;
283                                 } else {
284                                         usage();
285                                         exit(2);
286                                 }
287                         }
288                         n++;
289                 }
290                 argv++;
291         }

293         /*
294          * Check if we founded a named selector on the cmdline.
295          */
296         if (uselflg) {
297                 if (nselflg) {
298                         (void) fprintf(stderr, gettext(
299                             "module:instance:name:statistic and "
300                             "-m -i -n -s are mutually exclusive"));
301                         usage();
302                         exit(2);
303                 } else {
304                         free(nselector);
305                 }
306         } else {
307                 list_insert_tail(&selector_list, nselector);
308         }

310         assert(!list_is_empty(&selector_list));

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

315         kc = kstat_open();
316         if (kc == NULL) {
317                 perror("kstat_open");
318                 exit(3);
319         }

321         if (count > 1) {
322                 if (signal(SIGCONT, cont_handler) == SIG_ERR) {
```



```
323                         (void) fprintf(stderr, gettext(
324                             "signal failed"));
325                         exit(3);
326                 }
327         }

329         period_n = (hrtime_t)interval * NANOSEC;
330         start_n = gethrtime();

332         while (count == -1 || count-- > 0) {
333                 ks_instances_read(kc);
334                 ks_instances_print();

336                 if (interval && count) {
337                         sleep_until(&start_n, period_n, infinite_cycles,
338                             &caught_cont);
339                         (void) kstat_chain_update(kc);
340                         (void) putchar('\n');
341                 }
342         }

344         (void) kstat_close(kc);

346         return (g_matched);
347 }

349 /*
350  * Print usage.
351  */
352 static void
353 usage(void)
354 {
355         (void) fprintf(stderr, gettext(
356             "Usage:\n"
357             "kstat [ -Cjlpq ] [ -T d|u ] [ -c class ]\n"
358             "      [ -m module ] [ -i instance ] [ -n name ] [ -s statistic ]\n"
359             "      [ interval [ count ] ]\n"
360             "kstat [ -Cjlpq ] [ -T d|u ] [ -c class ]\n"
361             "      [ module:instance:name:statistic ... ]\n"
362             "      [ interval [ count ] ]\n"));
363 }

365 /*
366  * Sort compare function.
367  */
368 static int
369 compare_instances(ks_instance_t *l_arg, ks_instance_t *r_arg)
370 {
371         int     rval;

373         rval = strcasecmp(l_arg->ks_module, r_arg->ks_module);
374         if (rval == 0) {
375                 if (l_arg->ks_instance == r_arg->ks_instance) {
376                         return (strcasecmp(l_arg->ks_name, r_arg->ks_name));
377                 } else if (l_arg->ks_instance < r_arg->ks_instance) {
378                         return (-1);
379                 } else {
380                         return (1);
381                 }
382         } else {
383                 return (rval);
384         }
385 }

387 /*
388  * Inserts an instance in the per selector list.
```

```
 */
static void
nvpair_insert(ks_instance_t *ksi, char *name, ks_value_t *value,
    uchar_t data_type)
{
        ks_nvpair_t     *instance;
        ks_nvpair_t     *tmp;

        instance = (ks_nvpair_t *)malloc(sizeof (ks_nvpair_t));
        if (instance == NULL) {
                perror("malloc");
                exit(3);
        }

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

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

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

/*
 * Allocates a new all-matching selector.
 */
static ks_selector_t *
new_selector(void)
{
        ks_selector_t   *selector;

        selector = (ks_selector_t *)malloc(sizeof (ks_selector_t));
        if (selector == NULL) {
                perror("malloc");
                exit(3);
        }

        list_link_init(&selector->ks_next);

        selector->ks_module = "*";
        selector->ks_instance = "*";
        selector->ks_name = "*";
        selector->ks_statistic = "*";

        return (selector);
}

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

        for (f = module, t = key; *f != '\0'; f++, t++) {
                while (*f != '\0' && isdigit(*f))
                        f++;
                *t = *f;
        }
        *t++ = ':';
```



```
        for (f = name; *f != '\0'; f++, t++) {
                while (*f != '\0' && isdigit(*f))
                        f++;
                *t = *f;
        }
        *t = '\0';

        while (ks_raw_lookup[n].fn != NULL) {
                if (strncmp(ks_raw_lookup[n].name, key, strlen(key)) == 0)
                        return (ks_raw_lookup[n].fn);
                n++;
        }

        return (0);
}

/*
 * Iterate over all kernel statistics and save matches.
 */
static void
ks_instances_read(kstat_ctl_t *kc)
{
        kstat_raw_reader_t save_raw = NULL;
        kid_t           id;
        ks_selector_t   *selector;
        ks_instance_t   *ksi;
        ks_instance_t   *tmp;
        kstat_t         *kp;
        boolean_t       skip;
        char            *ks_number;

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

                /* Don't bother storing raw stats we don't understand */
                if (kp->ks_type == KSTAT_TYPE_RAW) {
                        save_raw = lookup_raw_kstat_fn(kp->ks_module,
                            kp->ks_name);
                        if (save_raw == NULL) {
#ifdef REPORT_UNKNOWN
                                (void) fprintf(stderr,
                                    "Unknown kstat type %s:%d:%s - "
                                    "%d of size %d\n", kp->ks_module,
                                    kp->ks_instance, kp->ks_name,
                                    kp->ks_ndata, kp->ks_data_size);
#endif
                                continue;
                        }
                }

                /*
                 * Iterate over the list of selectors and skip
                 * instances we dont want. We filter for statistics
                 * later, as we dont know them yet.
                 */
                skip = B_FALSE;
                (void) asprintf(&ks_number, "%d", kp->ks_instance);
                selector = list_head(&selector_list);
                while (selector != NULL) {
                        if (!(gmatch(kp->ks_module, selector->ks_module) != 0 &&
                            gmatch(ks_number, selector->ks_instance) != 0 &&
                            gmatch(kp->ks_name, selector->ks_name) != 0 &&
```

```
521                         gmatch(kp->ks_class, g_ks_class))) {
522                             skip = B_TRUE;
523                     }
524                     selector = list_next(&selector_list, selector);
525             }

527             free(ks_number);

529             if (skip) {
530                     continue;
531             }

533             /*
534              * Allocate a new instance and fill in the values
535              * we know so far.
536              */
537             ksi = (ks_instance_t *)malloc(sizeof (ks_instance_t));
538             if (ksi == NULL) {
539                     perror("malloc");
540                     exit(3);
541             }

543             list_link_init(&ksi->ks_next);

545             (void) strlcpy(ksi->ks_module, kp->ks_module, KSTAT_STRLEN);
546             (void) strlcpy(ksi->ks_name, kp->ks_name, KSTAT_STRLEN);
547             (void) strlcpy(ksi->ks_class, kp->ks_class, KSTAT_STRLEN);

549             ksi->ks_instance = kp->ks_instance;
550             ksi->ks_snaptime = kp->ks_snaptime;
551             ksi->ks_type = kp->ks_type;

553             list_create(&ksi->ks_nvlist, sizeof (ks_nvpair_t),
554                 offsetof(ks_nvpair_t, nv_next));

556             SAVE_HRTIME_X(ksi, "crtime", kp->ks_crtime);
557             SAVE_HRTIME_X(ksi, "snaptime", kp->ks_snaptime);
558             if (g_pflg) {
559                     SAVE_STRING_X(ksi, "class", kp->ks_class);
560             }

562             /* Insert this instance into a sorted list */
563             tmp = list_head(&instances_list);
564             while (tmp != NULL && compare_instances(ksi, tmp) > 0)
565                     tmp = list_next(&instances_list, tmp);

567             list_insert_before(&instances_list, tmp, ksi);

569             /* Read the actual statistics */
570             id = kstat_read(kc, kp, NULL);
571             if (id == -1) {
572                     perror("kstat_read");
573                     continue;
574             }

576             switch (kp->ks_type) {
577             case KSTAT_TYPE_RAW:
578                     save_raw(kp, ksi);
579                     break;
580             case KSTAT_TYPE_NAMED:
581                     save_named(kp, ksi);
582                     break;
583             case KSTAT_TYPE_INTR:
584                     save_intr(kp, ksi);
585                     break;
586             case KSTAT_TYPE_IO:
```



```
587                     save_io(kp, ksi);
588                     break;
589             case KSTAT_TYPE_TIMER:
590                     save_timer(kp, ksi);
591                     break;
592             default:
593                     assert(B_FALSE); /* Invalid type */
594                     break;
595             }
596     }
597 }

599 /*
600  * Print the value of a name-value pair.
601  */
602 static void
603 ks_value_print(ks_nvpair_t *nvpair)
604 {
605         switch (nvpair->data_type) {
606         case KSTAT_DATA_CHAR:
607                 (void) fprintf(stdout, "%s", nvpair->value.c);
608                 break;
609         case KSTAT_DATA_INT32:
610                 (void) fprintf(stdout, "%d", nvpair->value.i32);
611                 break;
612         case KSTAT_DATA_UINT32:
613                 (void) fprintf(stdout, "%u", nvpair->value.ui32);
614                 break;
615         case KSTAT_DATA_INT64:
616                 (void) fprintf(stdout, "%lld", nvpair->value.i64);
617                 break;
618         case KSTAT_DATA_UINT64:
619                 (void) fprintf(stdout, "%llu", nvpair->value.ui64);
620                 break;
621         case KSTAT_DATA_STRING:
622                 (void) fprintf(stdout, "%s", KSTAT_NAMED_STR_PTR(nvpair));
623                 break;
624         case KSTAT_DATA_HRTIME:
625                 if (nvpair->value.ui64 == 0)
626                         (void) fprintf(stdout, "0");
627                 else
628                         (void) fprintf(stdout, "%.9f",
629                             nvpair->value.ui64 / 1000000000.0);
630                 break;
631         default:
632                 assert(B_FALSE);
633         }
634 }

636 /*
637  * Print a single instance.
638  */
639 static void
640 ks_instance_print(ks_instance_t *ksi, ks_nvpair_t *nvpair)
641 {
642         if (g_headerflg) {
643                 if (!g_pflg) {
644                         (void) fprintf(stdout, DFLT_FMT,
645                             ksi->ks_module, ksi->ks_instance,
646                             ksi->ks_name, ksi->ks_class);
647                 }
648                 g_headerflg = B_FALSE;
649         }

651         if (g_pflg) {
652                 (void) fprintf(stdout, KS_PFMT,
```

```
653                     ksi->ks_module, ksi->ks_instance,
654                     ksi->ks_name, nvpair->name);
655                 if (!g_lflg) {
656                         (void) putchar(g_cflg ? ':': '\t');
657                         ks_value_print(nvpair);
658                 }
659         } else {
660                 (void) fprintf(stdout, KS_DFMT, nvpair->name);
661                 ks_value_print(nvpair);
662         }

664         (void) putchar('\n');
665 }

667 /*
668  * Print a single instance in JSON format.
669  */
670 static void
671 ks_instance_print_json(ks_instance_t *ksi, ks_nvpair_t *nvpair)
672 {
673         if (g_headerflg) {
674                 (void) fprintf(stdout, JSON_FMT,
675                     ksi->ks_module, ksi->ks_instance,
676                     ksi->ks_name, ksi->ks_class,
677                     ksi->ks_type);

679                 if (ksi->ks_snaptime == 0)
680                         (void) fprintf(stdout, "\t\"snaptime\": 0,\n");
681                 else
682                         (void) fprintf(stdout, "\t\"snaptime\": %.9f,\n",
683                             ksi->ks_snaptime / 1000000000.0);

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

687                 g_headerflg = B_FALSE;
688         }

690         (void) fprintf(stdout, KS_JFMT, nvpair->name);
691         if (nvpair->data_type == KSTAT_DATA_STRING) {
692                 (void) putchar('\"');
693                 ks_value_print(nvpair);
694                 (void) putchar('\"');
695         } else {
696                 ks_value_print(nvpair);
697         }
698         if (nvpair != list_tail(&ksi->ks_nvlist))
699                 (void) putchar(',');

701         (void) putchar('\n');
702 }

704 /*
705  * Print all instances.
706  */
707 static void
708 ks_instances_print(void)
709 {
710         ks_selector_t   *selector;
711         ks_instance_t   *ksi, *ktmp;
712         ks_nvpair_t     *nvpair, *ntmp;
713         void            (*ks_print_fn)(ks_instance_t *, ks_nvpair_t *);

715         if (g_timestamp_fmt != NODATE)
716                 print_timestamp(g_timestamp_fmt);

718         if (g_jflg) {
```



```
719                 ks_print_fn = &ks_instance_print_json;
720                 (void) putchar('[');
721         } else {
722                 ks_print_fn = &ks_instance_print;
723         }

725         /* Iterate over each selector */
726         selector = list_head(&selector_list);
727         while (selector != NULL) {

729                 /* Iterate over each instance */
730                 for (ksi = list_head(&instances_list); ksi != NULL;
731                     ksi = list_next(&instances_list, ksi)) {

733                         /* Finally iterate over each statistic */
734                         g_headerflg = B_TRUE;
735                         for (nvpair = list_head(&ksi->ks_nvlist);
736                             nvpair != NULL;
737                             nvpair = list_next(&ksi->ks_nvlist, nvpair)) {
738                                 if (gmatch(nvpair->name,
739                                     selector->ks_statistic) == 0)
740                                         continue;

742                                 g_matched = 0;
743                                 if (!g_qflg)
744                                         (*ks_print_fn)(ksi, nvpair);
745                         }

747                         if (!g_headerflg) {
748                                 if (g_jflg) {
749                                         (void) fprintf(stdout, "\t}\n}");
750                                         if (ksi != list_tail(&instances_list))
751                                                 (void) putchar(',');
752                                 } else if (!g_pflg) {
753                                         (void) putchar('\n');
754                                 }
755                         }
756                 }

758                 selector = list_next(&selector_list, selector);
759         }

761         if (g_jflg)
762                 (void) fprintf(stdout, "]\n");

764         (void) fflush(stdout);

766         /* Free the instances list */
767         ksi = list_head(&instances_list);
768         while (ksi != NULL) {
769                 nvpair = list_head(&ksi->ks_nvlist);
770                 while (nvpair != NULL) {
771                         ntmp = nvpair;
772                         nvpair = list_next(&ksi->ks_nvlist, nvpair);
773                         list_remove(&ksi->ks_nvlist, ntmp);
774                         if (ntmp->data_type == KSTAT_DATA_STRING)
775                                 free(ntmp->value.str.addr.ptr);
776                         free(ntmp);
777                 }

779                 ktmp = ksi;
780                 ksi = list_next(&instances_list, ksi);
781                 list_remove(&instances_list, ktmp);
782                 list_destroy(&ktmp->ks_nvlist);
783                 free(ktmp);
784         }
```

```
785 }

787 static void
788 save_cpu_stat(kstat_t *kp, ks_instance_t *ksi)
789 {
790         cpu_stat_t      *stat;
791         cpu_sysinfo_t   *sysinfo;
792         cpu_syswait_t   *syswait;
793         cpu_vminfo_t    *vminfo;

795         stat = (cpu_stat_t *)(kp->ks_data);
796         sysinfo = &stat->cpu_sysinfo;
797         syswait = &stat->cpu_syswait;
798         vminfo  = &stat->cpu_vminfo;

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



```
851         SAVE_UINT32(ksi, sysinfo, modunload);
852         SAVE_UINT32(ksi, sysinfo, bawrite);
853 #ifdef  STATISTICS      /* see header file */
854         SAVE_UINT32(ksi, sysinfo, rw_enters);
855         SAVE_UINT32(ksi, sysinfo, win_uo_cnt);
856         SAVE_UINT32(ksi, sysinfo, win_uu_cnt);
857         SAVE_UINT32(ksi, sysinfo, win_so_cnt);
858         SAVE_UINT32(ksi, sysinfo, win_su_cnt);
859         SAVE_UINT32(ksi, sysinfo, win_suo_cnt);
860 #endif

862         SAVE_INT32(ksi, syswait, iowait);
863         SAVE_INT32(ksi, syswait, swap);
864         SAVE_INT32(ksi, syswait, physio);

866         SAVE_UINT32(ksi, vminfo, pgrec);
867         SAVE_UINT32(ksi, vminfo, pgfrec);
868         SAVE_UINT32(ksi, vminfo, pgin);
869         SAVE_UINT32(ksi, vminfo, pgpgin);
870         SAVE_UINT32(ksi, vminfo, pgout);
871         SAVE_UINT32(ksi, vminfo, pgpgout);
872         SAVE_UINT32(ksi, vminfo, swapin);
873         SAVE_UINT32(ksi, vminfo, pgswapin);
874         SAVE_UINT32(ksi, vminfo, swapout);
875         SAVE_UINT32(ksi, vminfo, pgswapout);
876         SAVE_UINT32(ksi, vminfo, zfod);
877         SAVE_UINT32(ksi, vminfo, dfree);
878         SAVE_UINT32(ksi, vminfo, scan);
879         SAVE_UINT32(ksi, vminfo, rev);
880         SAVE_UINT32(ksi, vminfo, hat_fault);
881         SAVE_UINT32(ksi, vminfo, as_fault);
882         SAVE_UINT32(ksi, vminfo, maj_fault);
883         SAVE_UINT32(ksi, vminfo, cow_fault);
884         SAVE_UINT32(ksi, vminfo, prot_fault);
885         SAVE_UINT32(ksi, vminfo, softlock);
886         SAVE_UINT32(ksi, vminfo, kernel_asflt);
887         SAVE_UINT32(ksi, vminfo, pgrrun);
888         SAVE_UINT32(ksi, vminfo, execpgin);
889         SAVE_UINT32(ksi, vminfo, execpgout);
890         SAVE_UINT32(ksi, vminfo, execfree);
891         SAVE_UINT32(ksi, vminfo, anonpgin);
892         SAVE_UINT32(ksi, vminfo, anonpgout);
893         SAVE_UINT32(ksi, vminfo, anonfree);
894         SAVE_UINT32(ksi, vminfo, fspgin);
895         SAVE_UINT32(ksi, vminfo, fspgout);
896         SAVE_UINT32(ksi, vminfo, fsfree);
897 }

899 static void
900 save_var(kstat_t *kp, ks_instance_t *ksi)
901 {
902         struct var      *var = (struct var *)(kp->ks_data);

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

906         SAVE_INT32(ksi, var, v_buf);
907         SAVE_INT32(ksi, var, v_call);
908         SAVE_INT32(ksi, var, v_proc);
909         SAVE_INT32(ksi, var, v_maxupttl);
910         SAVE_INT32(ksi, var, v_nglobpris);
911         SAVE_INT32(ksi, var, v_maxsyspri);
912         SAVE_INT32(ksi, var, v_clist);
913         SAVE_INT32(ksi, var, v_maxup);
914         SAVE_INT32(ksi, var, v_hbuf);
915         SAVE_INT32(ksi, var, v_hmask);
916         SAVE_INT32(ksi, var, v_pbuf);
```

```
917         SAVE_INT32(ksi, var, v_sptmap);
918         SAVE_INT32(ksi, var, v_maxpmem);
919         SAVE_INT32(ksi, var, v_autoup);
920         SAVE_INT32(ksi, var, v_bufhwm);
921 }

923 static void
924 save_ncstats(kstat_t *kp, ks_instance_t *ksi)
925 {
926         struct ncstats  *ncstats = (struct ncstats *)(kp->ks_data);

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

930         SAVE_INT32(ksi, ncstats, hits);
931         SAVE_INT32(ksi, ncstats, misses);
932         SAVE_INT32(ksi, ncstats, enters);
933         SAVE_INT32(ksi, ncstats, dbl_enters);
934         SAVE_INT32(ksi, ncstats, long_enter);
935         SAVE_INT32(ksi, ncstats, long_look);
936         SAVE_INT32(ksi, ncstats, move_to_front);
937         SAVE_INT32(ksi, ncstats, purges);
938 }

940 static void
941 save_sysinfo(kstat_t *kp, ks_instance_t *ksi)
942 {
943         sysinfo_t       *sysinfo = (sysinfo_t *)(kp->ks_data);

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

947         SAVE_UINT32(ksi, sysinfo, updates);
948         SAVE_UINT32(ksi, sysinfo, runque);
949         SAVE_UINT32(ksi, sysinfo, runocc);
950         SAVE_UINT32(ksi, sysinfo, swpque);
951         SAVE_UINT32(ksi, sysinfo, swpocc);
952         SAVE_UINT32(ksi, sysinfo, waiting);
953 }

955 static void
956 save_vminfo(kstat_t *kp, ks_instance_t *ksi)
957 {
958         vminfo_t        *vminfo = (vminfo_t *)(kp->ks_data);

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

962         SAVE_UINT64(ksi, vminfo, freemem);
963         SAVE_UINT64(ksi, vminfo, swap_resv);
964         SAVE_UINT64(ksi, vminfo, swap_alloc);
965         SAVE_UINT64(ksi, vminfo, swap_avail);
966         SAVE_UINT64(ksi, vminfo, swap_free);
967         SAVE_UINT64(ksi, vminfo, updates);
968 }

970 static void
971 save_nfs(kstat_t *kp, ks_instance_t *ksi)
972 {
973         struct mntinfo_kstat *mntinfo = (struct mntinfo_kstat *)(kp->ks_data);

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

977         SAVE_STRING(ksi, mntinfo, mik_proto);
978         SAVE_UINT32(ksi, mntinfo, mik_vers);
979         SAVE_UINT32(ksi, mntinfo, mik_flags);
980         SAVE_UINT32(ksi, mntinfo, mik_secmod);
981         SAVE_UINT32(ksi, mntinfo, mik_curread);
982         SAVE_UINT32(ksi, mntinfo, mik_curwrite);
```



```
983         SAVE_INT32(ksi, mntinfo, mik_timeo);
984         SAVE_INT32(ksi, mntinfo, mik_retrans);
985         SAVE_UINT32(ksi, mntinfo, mik_acregmin);
986         SAVE_UINT32(ksi, mntinfo, mik_acregmax);
987         SAVE_UINT32(ksi, mntinfo, mik_acdirmin);
988         SAVE_UINT32(ksi, mntinfo, mik_acdirmax);
989         SAVE_UINT32_X(ksi, "lookup_srtt", mntinfo->mik_timers[0].srtt);
990         SAVE_UINT32_X(ksi, "lookup_deviate", mntinfo->mik_timers[0].deviate);
991         SAVE_UINT32_X(ksi, "lookup_rtxcur", mntinfo->mik_timers[0].rtxcur);
992         SAVE_UINT32_X(ksi, "read_srtt", mntinfo->mik_timers[1].srtt);
993         SAVE_UINT32_X(ksi, "read_deviate", mntinfo->mik_timers[1].deviate);
994         SAVE_UINT32_X(ksi, "read_rtxcur", mntinfo->mik_timers[1].rtxcur);
995         SAVE_UINT32_X(ksi, "write_srtt", mntinfo->mik_timers[2].srtt);
996         SAVE_UINT32_X(ksi, "write_deviate", mntinfo->mik_timers[2].deviate);
997         SAVE_UINT32_X(ksi, "write_rtxcur", mntinfo->mik_timers[2].rtxcur);
998         SAVE_UINT32(ksi, mntinfo, mik_noresponse);
999         SAVE_UINT32(ksi, mntinfo, mik_failover);
1000        SAVE_UINT32(ksi, mntinfo, mik_remap);
1001        SAVE_STRING(ksi, mntinfo, mik_curserver);
1002 }

1004 #ifdef __sparc
1005 static void
1006 save_sfmmu_global_stat(kstat_t *kp, ks_instance_t *ksi)
1007 {
1008        struct sfmmu_global_stat *sfmmug =
1009            (struct sfmmu_global_stat *)(kp->ks_data);

1011        assert(kp->ks_data_size == sizeof (struct sfmmu_global_stat));

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

```
1049         SAVE_INT32(ksi, sfmmug, sf_put_free_fail);
1050         SAVE_INT32(ksi, sfmmug, sf_pgcolor_conflict);
1051         SAVE_INT32(ksi, sfmmug, sf_uncache_conflict);
1052         SAVE_INT32(ksi, sfmmug, sf_unload_conflict);
1053         SAVE_INT32(ksi, sfmmug, sf_ism_uncache);
1054         SAVE_INT32(ksi, sfmmug, sf_ism_recache);
1055         SAVE_INT32(ksi, sfmmug, sf_recache);
1056         SAVE_INT32(ksi, sfmmug, sf_steal_count);
1057         SAVE_INT32(ksi, sfmmug, sf_pagesync);
1058         SAVE_INT32(ksi, sfmmug, sf_clrwrt);
1059         SAVE_INT32(ksi, sfmmug, sf_pagesync_invalid);
1060         SAVE_INT32(ksi, sfmmug, sf_kernel_xcalls);
1061         SAVE_INT32(ksi, sfmmug, sf_user_xcalls);
1062         SAVE_INT32(ksi, sfmmug, sf_tsb_grow);
1063         SAVE_INT32(ksi, sfmmug, sf_tsb_shrink);
1064         SAVE_INT32(ksi, sfmmug, sf_tsb_resize_failures);
1065         SAVE_INT32(ksi, sfmmug, sf_tsb_reloc);
1066         SAVE_INT32(ksi, sfmmug, sf_user_vtop);
1067         SAVE_INT32(ksi, sfmmug, sf_ctx_inv);
1068         SAVE_INT32(ksi, sfmmug, sf_tlb_reprog_pgsz);
1069         SAVE_INT32(ksi, sfmmug, sf_region_remap_demap);
1070         SAVE_INT32(ksi, sfmmug, sf_create_scd);
1071         SAVE_INT32(ksi, sfmmug, sf_join_scd);
1072         SAVE_INT32(ksi, sfmmug, sf_leave_scd);
1073         SAVE_INT32(ksi, sfmmug, sf_destroy_scd);
1074 }
1075 #endif

1077 #ifdef __sparc
1078 static void
1079 save_sfmmu_tsbsize_stat(kstat_t *kp, ks_instance_t *ksi)
1080 {
1081         struct sfmmu_tsbsize_stat *sfmmut;

1083         assert(kp->ks_data_size == sizeof (struct sfmmu_tsbsize_stat));
1084         sfmmut = (struct sfmmu_tsbsize_stat *)(kp->ks_data);

1086         SAVE_INT32(ksi, sfmmut, sf_tsbsz_8k);
1087         SAVE_INT32(ksi, sfmmut, sf_tsbsz_16k);
1088         SAVE_INT32(ksi, sfmmut, sf_tsbsz_32k);
1089         SAVE_INT32(ksi, sfmmut, sf_tsbsz_64k);
1090         SAVE_INT32(ksi, sfmmut, sf_tsbsz_128k);
1091         SAVE_INT32(ksi, sfmmut, sf_tsbsz_256k);
1092         SAVE_INT32(ksi, sfmmut, sf_tsbsz_512k);
1093         SAVE_INT32(ksi, sfmmut, sf_tsbsz_1m);
1094         SAVE_INT32(ksi, sfmmut, sf_tsbsz_2m);
1095         SAVE_INT32(ksi, sfmmut, sf_tsbsz_4m);
1096 }
1097 #endif

1099 #ifdef __sparc
1100 static void
1101 save_simmstat(kstat_t *kp, ks_instance_t *ksi)
1102 {
1103         uchar_t *simmstat;
1104         char    *simm_buf;
1105         char    *list = NULL;
1106         int     i;

1108         assert(kp->ks_data_size == sizeof (uchar_t) * SIMM_COUNT);

1110         for (i = 0, simmstat = (uchar_t *)(kp->ks_data); i < SIMM_COUNT - 1;
1111             i++, simmstat++) {
1112                 if (list == NULL) {
1113                         (void) asprintf(&simm_buf, "%d,", *simmstat);
1114                 } else {
```



```
1115                         (void) asprintf(&simm_buf, "%s%d,", list, *simmstat);
1116                         free(list);
1117                 }
1118                 list = simm_buf;
1119         }

1121         (void) asprintf(&simm_buf, "%s%d", list, *simmstat);
1122         SAVE_STRING_X(ksi, "status", simm_buf);
1123         free(list);
1124         free(simm_buf);
1125 }
1126 #endif

1128 #ifdef __sparc
1129 /*
1130  * Helper function for save_temperature().
1131  */
1132 static char *
1133 short_array_to_string(short *shortp, int len)
1134 {
1135         char    *list = NULL;
1136         char    *list_buf;

1138         for (; len > 1; len--, shortp++) {
1139                 if (list == NULL) {
1140                         (void) asprintf(&list_buf, "%d,", *shortp);
1141                 } else {
1142                         (void) asprintf(&list_buf, "%s%d,", list, *shortp);
1143                         free(list);
1144                 }
1145                 list = list_buf;
1146         }

1148         (void) asprintf(&list_buf, "%s%s", list, *shortp);
1149         free(list);
1150         return (list_buf);
1151 }

1153 static void
1154 save_temperature(kstat_t *kp, ks_instance_t *ksi)
1155 {
1156         struct temp_stats *temps = (struct temp_stats *)(kp->ks_data);
1157         char    *buf;
1158         int     n = 1;

1160         assert(kp->ks_data_size == sizeof (struct temp_stats));

1162         SAVE_UINT32(ksi, temps, index);

1164         buf = short_array_to_string(temps->l1, L1_SZ);
1165         SAVE_STRING_X(ksi, "l1", buf);
1166         free(buf);

1168         buf = short_array_to_string(temps->l2, L2_SZ);
1169         SAVE_STRING_X(ksi, "l2", buf);
1170         free(buf);

1172         buf = short_array_to_string(temps->l3, L3_SZ);
1173         SAVE_STRING_X(ksi, "l3", buf);
1174         free(buf);

1176         buf = short_array_to_string(temps->l4, L4_SZ);
1177         SAVE_STRING_X(ksi, "l4", buf);
1178         free(buf);

1180         buf = short_array_to_string(temps->l5, L5_SZ);
```

```
1181         SAVE_STRING_X(ksi, "l5", buf);
1182         free(buf);

1184         SAVE_INT32(ksi, temps, max);
1185         SAVE_INT32(ksi, temps, min);
1186         SAVE_INT32(ksi, temps, state);
1187         SAVE_INT32(ksi, temps, temp_cnt);
1188         SAVE_INT32(ksi, temps, shutdown_cnt);
1189         SAVE_INT32(ksi, temps, version);
1190         SAVE_INT32(ksi, temps, trend);
1191         SAVE_INT32(ksi, temps, override);
1192 }
1193 #endif

1195 #ifdef __sparc
1196 static void
1197 save_temp_over(kstat_t *kp, ks_instance_t *ksi)
1198 {
1199         short   *sh = (short *)(kp->ks_data);
1200         char    *value;

1202         assert(kp->ks_data_size == sizeof (short));

1204         (void) asprintf(&value, "%hu", *sh);
1205         SAVE_STRING_X(ksi, "override", value);
1206         free(value);
1207 }
1208 #endif

1210 #ifdef __sparc
1211 static void
1212 save_ps_shadow(kstat_t *kp, ks_instance_t *ksi)
1213 {
1214         uchar_t *uchar = (uchar_t *)(kp->ks_data);

1216         assert(kp->ks_data_size == SYS_PS_COUNT);

1218         SAVE_CHAR_X(ksi, "core_0", *uchar++);
1219         SAVE_CHAR_X(ksi, "core_1", *uchar++);
1220         SAVE_CHAR_X(ksi, "core_2", *uchar++);
1221         SAVE_CHAR_X(ksi, "core_3", *uchar++);
1222         SAVE_CHAR_X(ksi, "core_4", *uchar++);
1223         SAVE_CHAR_X(ksi, "core_5", *uchar++);
1224         SAVE_CHAR_X(ksi, "core_6", *uchar++);
1225         SAVE_CHAR_X(ksi, "core_7", *uchar++);
1226         SAVE_CHAR_X(ksi, "pps_0", *uchar++);
1227         SAVE_CHAR_X(ksi, "clk_33", *uchar++);
1228         SAVE_CHAR_X(ksi, "clk_50", *uchar++);
1229         SAVE_CHAR_X(ksi, "v5_p", *uchar++);
1230         SAVE_CHAR_X(ksi, "v12_p", *uchar++);
1231         SAVE_CHAR_X(ksi, "v5_aux", *uchar++);
1232         SAVE_CHAR_X(ksi, "v5_p_pch", *uchar++);
1233         SAVE_CHAR_X(ksi, "v12_p_pch", *uchar++);
1234         SAVE_CHAR_X(ksi, "v3_pch", *uchar++);
1235         SAVE_CHAR_X(ksi, "v5_pch", *uchar++);
1236         SAVE_CHAR_X(ksi, "p_fan", *uchar++);
1237 }
1238 #endif

1240 #ifdef __sparc
1241 static void
1242 save_fault_list(kstat_t *kp, ks_instance_t *ksi)
1243 {
1244         struct ft_list *fault;
1245         char    name[KSTAT_STRLEN + 7];
1246         int     i;
```



```
1248         for (i = 1, fault = (struct ft_list *)(kp->ks_data);
1249             i <= 999999 && i <= kp->ks_data_size / sizeof (struct ft_list);
1250             i++, fault++) {
1251                 (void) snprintf(name, sizeof (name), "unit_%d", i);
1252                 SAVE_INT32_X(ksi, name, fault->unit);
1253                 (void) snprintf(name, sizeof (name), "type_%d", i);
1254                 SAVE_INT32_X(ksi, name, fault->type);
1255                 (void) snprintf(name, sizeof (name), "fclass_%d", i);
1256                 SAVE_INT32_X(ksi, name, fault->fclass);
1257                 (void) snprintf(name, sizeof (name), "create_time_%d", i);
1258                 SAVE_HRTIME_X(ksi, name, fault->create_time);
1259                 (void) snprintf(name, sizeof (name), "msg_%d", i);
1260                 SAVE_STRING_X(ksi, name, faultp->msg);
1261         }
1262 }
1263 #endif

1265 static void
1266 save_named(kstat_t *kp, ks_instance_t *ksi)
1267 {
1268         kstat_named_t *knp;
1269         int     n;

1271         for (n = kp->ks_ndata, knp = KSTAT_NAMED_PTR(kp); n > 0; n--, knp++) {
1272                 switch (knp->data_type) {
1273                 case KSTAT_DATA_CHAR:
1274                         nvpair_insert(ksi, knp->name,
1275                             (ks_value_t *)&knp->value, KSTAT_DATA_CHAR);
1276                         break;
1277                 case KSTAT_DATA_INT32:
1278                         nvpair_insert(ksi, knp->name,
1279                             (ks_value_t *)&knp->value, KSTAT_DATA_INT32);
1280                         break;
1281                 case KSTAT_DATA_UINT32:
1282                         nvpair_insert(ksi, knp->name,
1283                             (ks_value_t *)&knp->value, KSTAT_DATA_UINT32);
1284                         break;
1285                 case KSTAT_DATA_INT64:
1286                         nvpair_insert(ksi, knp->name,
1287                             (ks_value_t *)&knp->value, KSTAT_DATA_INT64);
1288                         break;
1289                 case KSTAT_DATA_UINT64:
1290                         nvpair_insert(ksi, knp->name,
1291                             (ks_value_t *)&knp->value, KSTAT_DATA_UINT64);
1292                         break;
1293                 case KSTAT_DATA_STRING:
1294                         SAVE_STRING_X(ksi, knp->name, KSTAT_NAMED_STR_PTR(knp));
1295                         break;
1296                 default:
1297                         assert(B_FALSE); /* Invalid data type */
1298                         break;
1299                 }
1300         }
1301 }

1303 static void
1304 save_intr(kstat_t *kp, ks_instance_t *ksi)
1305 {
1306         kstat_intr_t *intr = KSTAT_INTR_PTR(kp);
1307         char    *intr_names[] = {"hard", "soft", "watchdog", "spurious",
1308             "multiple_service"};
1309         int     n;

1311         for (n = 0; n < KSTAT_NUM_INTRS; n++)
1312                 SAVE_UINT32_X(ksi, intr_names[n], intr->intrs[n]);
```

```
1313 }

1315 static void
1316 save_io(kstat_t *kp, ks_instance_t *ksi)
1317 {
1318         kstat_io_t      *ksio = KSTAT_IO_PTR(kp);

1320         SAVE_UINT64(ksi, ksio, nread);
1321         SAVE_UINT64(ksi, ksio, nwritten);
1322         SAVE_UINT32(ksi, ksio, reads);
1323         SAVE_UINT32(ksi, ksio, writes);
1324         SAVE_HRTIME(ksi, ksio, wtime);
1325         SAVE_HRTIME(ksi, ksio, wlentime);
1326         SAVE_HRTIME(ksi, ksio, wlastupdate);
1327         SAVE_HRTIME(ksi, ksio, rtime);
1328         SAVE_HRTIME(ksi, ksio, rlentime);
1329         SAVE_HRTIME(ksi, ksio, rlastupdate);
1330         SAVE_UINT32(ksi, ksio, wcnt);
1331         SAVE_UINT32(ksi, ksio, rcnt);
1332 }

1334 static void
1335 save_timer(kstat_t *kp, ks_instance_t *ksi)
1336 {
1337         kstat_timer_t   *ktimer = KSTAT_TIMER_PTR(kp);

1339         SAVE_STRING(ksi, ktimer, name);
1340         SAVE_UINT64(ksi, ktimer, num_events);
1341         SAVE_HRTIME(ksi, ktimer, elapsed_time);
1342         SAVE_HRTIME(ksi, ktimer, min_time);
1343         SAVE_HRTIME(ksi, ktimer, max_time);
1344         SAVE_HRTIME(ksi, ktimer, start_time);
1345         SAVE_HRTIME(ksi, ktimer, stop_time);
1346 }
1347 #endif /* ! codereview */
```

```
**********************************************************
    6701 Wed Nov 28 23:08:58 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>
kstat
**********************************************************
```

```
/*
 * 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 Wed Nov 28 23:08:58 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>
kstat
**********************************************************
```

```
'\" 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
```

```
248 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.
```