```
**********************************************************
   86490 Wed Jan  1 08:36:45 2014
new/usr/src/cmd/cron/cron.c
4378 Clean up %C in *time() functions
438 need documentation for strftime %s flag
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */
  21 /*
  22  * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
  23  * Use is subject to license terms.
  24  *
  25  * Copyright 2013 Joshua M. Clulow <josh@sysmgr.org>
  26  *
  27  * Copyright (c) 2014 Gary Mills
  28  */

  30 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
  31 /*        All Rights Reserved   */

  33 /*      Copyright (c) 1987, 1988 Microsoft Corporation  */
  34 /*        All Rights Reserved   */

  36 #ifdef lint
  37 /* make lint happy */
  38 #define __EXTENSIONS__
  39 #endif

  41 #include <sys/contract/process.h>
  42 #include <sys/ctfs.h>
  43 #include <sys/param.h>
  44 #include <sys/resource.h>
  45 #include <sys/stat.h>
  46 #include <sys/task.h>
  47 #include <sys/time.h>
  48 #include <sys/types.h>
  49 #include <sys/utsname.h>
  50 #include <sys/wait.h>

  52 #include <security/pam_appl.h>

  54 #include <alloca.h>
  55 #include <ctype.h>
  56 #include <deflt.h>
  57 #include <dirent.h>
  58 #include <errno.h>
  59 #include <fcntl.h>
  60 #include <grp.h>
```



```
  61 #include <libcontract.h>
  62 #include <libcontract_priv.h>
  63 #include <limits.h>
  64 #include <locale.h>
  65 #include <poll.h>
  66 #include <project.h>
  67 #include <pwd.h>
  68 #include <signal.h>
  69 #include <stdarg.h>
  70 #include <stdio.h>
  71 #include <stdlib.h>
  72 #include <string.h>
  73 #include <stropts.h>
  74 #include <time.h>
  75 #include <unistd.h>
  76 #include <libzoneinfo.h>

  78 #include "cron.h"

  80 /*
  81  * #define      DEBUG
  82  */

  84 #define MAIL            "/usr/bin/mail" /* mail program to use */
  85 #define CONSOLE         "/dev/console"  /* where messages go when cron dies */

  87 #define TMPINFILE       "/tmp/crinXXXXXX"  /* file to put stdin in for cmd  */
  88 #define TMPDIR          "/tmp"
  89 #define PFX             "crout"
  90 #define TMPOUTFILE      "/tmp/croutXXXXXX" /* file to place stdout, stderr */

  92 #define INMODE          00400           /* mode for stdin file  */
  93 #define OUTMODE         00600           /* mode for stdout file */
  94 #define ISUID           S_ISUID         /* mode for verifing at jobs */

  96 #define INFINITY        2147483647L     /* upper bound on time  */
  97 #define CUSHION         180L
  98 #define ZOMB            100             /* proc slot used for mailing output */

 100 #define JOBF            'j'
 101 #define NICEF           'n'
 102 #define USERF           'u'
 103 #define WAITF           'w'

 105 #define BCHAR           '>'
 106 #define ECHAR           '<'

 108 #define DEFAULT         0
 109 #define LOAD            1
 110 #define QBUFSIZ         80

 112 /* Defined actions for crabort() routine */
 113 #define NO_ACTION       000
 114 #define REMOVE_FIFO     001
 115 #define CONSOLE_MSG     002

 117 #define BADCD           "can't change directory to the crontab directory."
 118 #define NOREADDIR       "can't read the crontab directory."

 120 #define BADJOBOPEN      "unable to read your at job."
 121 #define BADSHELL        "because your login shell \
 122 isn't /usr/bin/sh, you can't use cron."

 124 #define BADSTAT         "can't access your crontab or at-job file. Resubmit it."
 125 #define BADPROJID       "can't set project id for your job."
 126 #define CANTCDHOME      "can't change directory to %s.\
```

```
 127 \nYour commands will not be executed."
 128 #define CANTEXECSH      "unable to exec the shell, %s, for one of your \
 129 commands."
 130 #define CANT_STR_LEN (sizeof (CANTEXECSH) > sizeof (CANTCDHOME) ? \
 131         sizeof (CANTEXECSH) : sizeof (CANTCDHOME))
 132 #define NOREAD          "can't read your crontab file.  Resubmit it."
 133 #define BADTYPE         "crontab or at-job file is not a regular file.\n"
 134 #define NOSTDIN         "unable to create a standard input file for \
 135 one of your crontab commands. \
 136 \nThat command was not executed."

 138 #define NOTALLOWED      "you are not authorized to use cron.  Sorry."
 139 #define STDERRMSG       "\n\n*****************************************\
 140 *****\nCron: The previous message is the \
 141 standard output and standard error \
 142 \nof one of your cron commands.\n"

 144 #define STDOUTERR       "one of your commands generated output or errors, \
 145 but cron was unable to mail you this output.\
 146 \nRemember to redirect standard output and standard \
 147 error for each of your commands."

 149 #define CLOCK_DRIFT     "clock time drifted backwards after event!\n"
 150 #define PIDERR          "unexpected pid returned %d (ignored)"
 151 #define CRONTABERR      "Subject: Your crontab file has an error in it\n\n"
 152 #define MALLOCERR       "out of space, cannot create new string\n"

 154 #define DIDFORK didfork
 155 #define NOFORK !didfork

 157 #define MAILBUFLEN      (8*1024)
 158 #define LINELIMIT       80
 159 #define MAILBINITFREE   (MAILBUFLEN - (sizeof (cte_intro) - 1) \
 160             - (sizeof (cte_trail1) - 1) - (sizeof (cte_trail2) - 1) - 1)

 162 #define ERR_CRONTABENT  0       /* error in crontab file entry */
 163 #define ERR_UNIXERR     1       /* error in some system call */
 164 #define ERR_CANTEXECCRON 2      /* error setting up "cron" job environment */
 165 #define ERR_CANTEXECAT  3       /* error setting up "at" job environment */
 166 #define ERR_NOTREG      4       /* error not a regular file */

 168 #define PROJECT         "project="

 170 #define MAX_LOST_CONTRACTS      2048    /* reset if this many failed abandons */

 172 #define FORMAT  "%a %b %e %H:%M:%S %Y"
 173 static char     timebuf[80];

 175 static struct message msgbuf;

 177 struct shared {
 178         int count;                      /* usage count */
 179         void (*free)(void *obj);        /* routine that will free obj */
 180         void *obj;                      /* object */
 181 };
_____unchanged_portion_omitted_

 288 extern char **environ;

 290 #define DEFTZ           "GMT"
 291 static  int     log = 0;
 292 static  char    hzname[10];

 294 static void cronend(int);
 295 static void thaw_handler(int);
 296 static void child_handler(int);
```



```
 297 static void child_sigreset(void);

 299 static void mod_ctab(char *, time_t);
 300 static void mod_atjob(char *, time_t);
 301 static void add_atevent(struct usr *, char *, time_t, int);
 302 static void rm_ctevents(struct usr *);
 303 static void cleanup(struct runinfo *rn, int r);
 304 static void crabort(char *, int);
 305 static void msg(char *fmt, ...);
 306 static void ignore_msg(char *, char *, struct event *);
 307 static void logit(int, struct runinfo *, int);
 308 static void parsqdef(char *);
 309 static void defaults();
 310 static void initialize(int);
 311 static void quedefs(int);
 312 static int idle(long);
 313 static struct usr *find_usr(char *);
 314 static int ex(struct event *e);
 315 static void read_dirs(int);
 316 static void mail(char *, char *, int);
 317 static char *next_field(int, int);
 318 static void readcron(struct usr *, time_t);
 319 static int next_ge(int, char *);
 320 static void free_if_unused(struct usr *);
 321 static void del_atjob(char *, char *);
 322 static void del_ctab(char *);
 323 static void resched(int);
 324 static int msg_wait(long);
 325 static struct runinfo *rinfo_get(pid_t);
 326 static void rinfo_free(struct runinfo *rp);
 327 static void mail_result(struct usr *p, struct runinfo *pr, size_t filesize);
 328 static time_t next_time(struct event *, time_t);
 329 static time_t get_switching_time(int, time_t);
 330 static time_t xmktime(struct tm *);
 331 static void process_msg(struct message *, time_t);
 332 static void reap_child(void);
 333 static void miscpid_insert(pid_t);
 334 static int miscpid_delete(pid_t);
 335 static void contract_set_template(void);
 336 static void contract_clear_template(void);
 337 static void contract_abandon_latest(pid_t);

 339 static void cte_init(void);
 340 static void cte_add(int, char *);
 341 static void cte_valid(void);
 342 static int cte_istoomany(void);
 343 static void cte_sendmail(char *);

 345 static int set_user_cred(const struct usr *, struct project *);

 347 static struct shared *create_shared_str(char *str);
 348 static struct shared *dup_shared(struct shared *obj);
 349 static void rel_shared(struct shared *obj);
 350 static void *get_obj(struct shared *obj);
 351 /*
 352  * last_time is set immediately prior to exection of an event (via ex())
 353  * to indicate the last time an event was executed.  This was (surely)
 354  * it's original intended use.
 355  */
 356 static time_t last_time, init_time, t_old;
 357 static int reset_needed; /* set to 1 when cron(1M) needs to re-initialize */

 359 static int              refresh;
 360 static sigset_t         defmask, sigmask;

 362 /*
```

```
 363  * BSM hooks
 364  */
 365 extern int      audit_cron_session(char *, char *, uid_t, gid_t, char *);
 366 extern void     audit_cron_new_job(char *, int, void *);
 367 extern void     audit_cron_bad_user(char *);
 368 extern void     audit_cron_user_acct_expired(char *);
 369 extern int      audit_cron_create_anc_file(char *, char *, char *, uid_t);
 370 extern int      audit_cron_delete_anc_file(char *, char *);
 371 extern int      audit_cron_is_anc_name(char *);
 372 extern int      audit_cron_mode();

 374 static int cron_conv(int, struct pam_message **,
 375                 struct pam_response **, void *);

 377 static struct pam_conv pam_conv = {cron_conv, NULL};
 378 static pam_handle_t *pamh;      /* Authentication handle */

 380 /*
 381  * Function to help check a user's credentials.
 382  */

 384 static int verify_user_cred(struct usr *u);

 386 /*
 387  * Values returned by verify_user_cred and set_user_cred:
 388  */

 390 #define VUC_OK          0
 391 #define VUC_BADUSER     1
 392 #define VUC_NOTINGROUP  2
 393 #define VUC_EXPIRED     3
 394 #define VUC_NEW_AUTH    4

 396 /*
 397  * Modes of process_anc_files function
 398  */
 399 #define CRON_ANC_DELETE 1
 400 #define CRON_ANC_CREATE 0

 402 /*
 403  * Functions to remove a user or job completely from the running database.
 404  */
 405 static void clean_out_atjobs(struct usr *u);
 406 static void clean_out_ctab(struct usr *u);
 407 static void clean_out_user(struct usr *u);
 408 static void cron_unlink(char *name);
 409 static void process_anc_files(int);

 411 /*
 412  * functions in elm.c
 413  */
 414 extern void el_init(int, time_t, time_t, int);
 415 extern int el_add(void *, time_t, int);
 416 extern void el_remove(int, int);
 417 extern int el_empty(void);
 418 extern void *el_first(void);
 419 extern void el_delete(void);

 421 static int valid_entry(char *, int);
 422 static struct usr *create_ulist(char *, int);
 423 static void init_cronevent(char *, int);
 424 static void init_atevent(char *, time_t, int, int);
 425 static void update_atevent(struct usr *, char *, time_t, int);

 427 int
 428 main(int argc, char *argv[])
```



```
 429 {
 430         time_t t;
 431         time_t ne_time;         /* amt of time until next event execution */
 432         time_t newtime, lastmtime = 0L;
 433         struct usr *u;
 434         struct event *e, *e2, *eprev;
 435         struct stat buf;
 436         pid_t rfork;
 437         struct sigaction act;

 439         /*
 440          * reset_needed is set to 1 whenever el_add() finds out that a cron
 441          * job is scheduled to be run before the time when cron(1M) daemon
 442          * initialized.
 443          * Other cases where a reset is needed is when ex() finds that the
 444          * event to be executed is being run at the wrong time, or when idle()
 445          * determines that time was reset.
 446          * We immediately return to the top of the while (TRUE) loop in
 447          * main() where the event list is cleared and rebuilt, and reset_needed
 448          * is set back to 0.
 449          */
 450         reset_needed = 0;

 452         /*
 453          * Only the privileged user can run this command.
 454          */
 455         if (getuid() != 0)
 456                 crabort(NOTALLOWED, 0);

 458 begin:
 459         (void) setlocale(LC_ALL, "");
 460         /* fork unless 'nofork' is specified */
 461         if ((argc <= 1) || (strcmp(argv[1], "nofork"))) {
 462                 if (rfork = fork()) {
 463                         if (rfork == (pid_t)-1) {
 464                                 (void) sleep(30);
 465                                 goto begin;
 466                         }
 467                         return (0);
 468                 }
 469                 didfork++;
 470                 (void) setpgrp();       /* detach cron from console */
 471         }

 473         (void) umask(022);
 474         (void) signal(SIGHUP, SIG_IGN);
 475         (void) signal(SIGINT, SIG_IGN);
 476         (void) signal(SIGQUIT, SIG_IGN);
 477         (void) signal(SIGTERM, cronend);

 479         defaults();
 480         initialize(1);
 481         quedefs(DEFAULT);       /* load default queue definitions */
 482         cron_pid = getpid();
 483         msg("*** cron started ***   pid = %d", cron_pid);

 485         /* setup THAW handler */
 486         act.sa_handler = thaw_handler;
 487         act.sa_flags = 0;
 488         (void) sigemptyset(&act.sa_mask);
 489         (void) sigaction(SIGTHAW, &act, NULL);

 491         /* setup CHLD handler */
 492         act.sa_handler = child_handler;
 493         act.sa_flags = 0;
 494         (void) sigemptyset(&act.sa_mask);
```

```
495         (void) sigaddset(&act.sa_mask, SIGCLD);
496         (void) sigaction(SIGCLD, &act, NULL);

498         (void) sigemptyset(&defmask);
499         (void) sigemptyset(&sigmask);
500         (void) sigaddset(&sigmask, SIGCLD);
501         (void) sigaddset(&sigmask, SIGTHAW);
502         (void) sigprocmask(SIG_BLOCK, &sigmask, NULL);

504         t_old = init_time;
505         last_time = t_old;
506         for (;;) {              /* MAIN LOOP */
507                 t = time(NULL);
508                 if ((t_old > t) || (t-last_time > CUSHION) || reset_needed) {
509                         reset_needed = 0;
510                         /*
511                          * the time was set backwards or forward or
512                          * refresh is requested.
513                          */
514                         if (refresh)
515                                 msg("re-scheduling jobs");
516                         else
517                                 msg("time was reset, re-initializing");
518                         el_delete();
519                         u = uhead;
520                         while (u != NULL) {
521                                 rm_ctevents(u);
522                                 e = u->atevents;
523                                 while (e != NULL) {
524                                         free(e->cmd);
525                                         e2 = e->link;
526                                         free(e);
527                                         e = e2;
528                                 }
529                                 u->atevents = NULL;
530                                 u = u->nextusr;
531                         }
532                         (void) close(msgfd);
533                         initialize(0);
534                         t = time(NULL);
535                         last_time = t;
536                         /*
537                          * reset_needed might have been set in the functions
538                          * call path from initialize()
539                          */
540                         if (reset_needed) {
541                                 continue;
542                         }
543                 }
544                 t_old = t;

546                 if (next_event == NULL && !el_empty()) {
547                         next_event = (struct event *)el_first();
548                 }
549                 if (next_event == NULL) {
550                         ne_time = INFINITY;
551                 } else {
552                         ne_time = next_event->time - t;
553 #ifdef DEBUG
554                         cftime(timebuf, "%+", &next_event->time);
552                         cftime(timebuf, "%C", &next_event->time);
555                         (void) fprintf(stderr, "next_time=%ld %s\n",
556                             next_event->time, timebuf);
557 #endif
558                 }
559                 if (ne_time > 0) {
```



```
560                         /*
561                          * reset_needed may be set in the functions call path
562                          * from idle()
563                          */
564                         if (idle(ne_time) || reset_needed) {
565                                 reset_needed = 1;
566                                 continue;
567                         }
568                 }

570                 if (stat(QUEDEFS, &buf)) {
571                         msg("cannot stat QUEDEFS file");
572                 } else if (lastmtime != buf.st_mtime) {
573                         quedefs(LOAD);
574                         lastmtime = buf.st_mtime;
575                 }

577                 last_time = next_event->time; /* save execution time */

579                 /*
580                  * reset_needed may be set in the functions call path
581                  * from ex()
582                  */
583                 if (ex(next_event) || reset_needed) {
584                         reset_needed = 1;
585                         continue;
586                 }

588                 switch (next_event->etype) {
589                 case CRONEVENT:
590                         /* add cronevent back into the main event list */
591                         if (delayed) {
592                                 delayed = 0;
593                                 break;
594                         }

596                         /*
597                          * check if time(0)< last_time. if so, then the
598                          * system clock has gone backwards. to prevent this
599                          * job from being started twice, we reschedule this
600                          * job for the >>next time after last_time<<, and
601                          * then set next_event->time to this. note that
602                          * crontab's resolution is 1 minute.
603                          */

605                         if (last_time > time(NULL)) {
606                                 msg(CLOCK_DRIFT);
607                                 /*
608                                  * bump up to next 30 second
609                                  * increment
610                                  * 1 <= newtime <= 30
611                                  */
612                                 newtime = 30 - (last_time % 30);
613                                 newtime += last_time;

615                                 /*
616                                  * get the next scheduled event,
617                                  * not the one that we just
618                                  * kicked off!
619                                  */
620                                 next_event->time =
621                                     next_time(next_event, newtime);
622                                 t_old = time(NULL);
623                         } else {
624                                 next_event->time =
625                                     next_time(next_event, (time_t)0);
```

```
                        }
#ifdef DEBUG
                        cftime(timebuf, "%+", &next_event->time);
                        cftime(timebuf, "%C", &next_event->time);
                        (void) fprintf(stderr,
                            "pushing back cron event %s at %ld (%s)\n",
                            next_event->cmd, next_event->time, timebuf);
#endif

                        switch (el_add(next_event, next_event->time,
                            (next_event->u)->ctid)) {
                        case -1:
                                ignore_msg("main", "cron", next_event);
                                break;
                        case -2: /* event time lower than init time */
                                reset_needed = 1;
                                break;
                        }
                        break;
                default:
                        /* remove at or batch job from system */
                        if (delayed) {
                                delayed = 0;
                                break;
                        }
                        eprev = NULL;
                        e = (next_event->u)->atevents;
                        while (e != NULL) {
                                if (e == next_event) {
                                        if (eprev == NULL)
                                                (e->u)->atevents = e->link;
                                        else
                                                eprev->link = e->link;
                                        free(e->cmd);
                                        free(e);
                                        break;
                                } else {
                                        eprev = e;
                                        e = e->link;
                                }
                        }
                        break;
                }
                next_event = NULL;
        }

        /*NOTREACHED*/
}
_____unchanged_portion_omitted_

static char line[CTLINESIZE];   /* holds a line from a crontab file */
static int cursor;              /* cursor for the above line */

static void
readcron(struct usr *u, time_t reftime)
{
        /*
         * readcron reads in a crontab file for a user (u). The list of
         * events for user u is built, and u->events is made to point to
         * this list. Each event is also entered into the main event
         * list.
         */
        FILE *cf;       /* cf will be a user's crontab file */
        struct event *e;
        int start;
        unsigned int i;
```



```
        char namebuf[PATH_MAX];
        char *pname;
        struct shared *tz = NULL;
        struct shared *home = NULL;
        struct shared *shell = NULL;
        int lineno = 0;

        /* read the crontab file */
        cte_init();             /* Init error handling */
        if (cwd != CRON) {
                if (snprintf(namebuf, sizeof (namebuf), "%s/%s",
                    CRONDIR, u->name) >= sizeof (namebuf)) {
                        return;
                }
                pname = namebuf;
        } else {
                pname = u->name;
        }
        if ((cf = fopen(pname, "r")) == NULL) {
                mail(u->name, NOREAD, ERR_UNIXERR);
                return;
        }
        while (fgets(line, CTLINESIZE, cf) != NULL) {
                char *tmp;
                /* process a line of a crontab file */
                lineno++;
                if (cte_istoomany())
                        break;
                cursor = 0;
                while (line[cursor] == ' ' || line[cursor] == '\t')
                        cursor++;
                if (line[cursor] == '#' || line[cursor] == '\n')
                        continue;

                if (strncmp(&line[cursor], ENV_TZ,
                    strlen(ENV_TZ)) == 0) {
                        if ((tmp = strchr(&line[cursor], '\n')) != NULL) {
                                *tmp = NULL;
                        }

                        if (!isvalid_tz(&line[cursor + strlen(ENV_TZ)], NULL,
                            _VTZ_ALL)) {
                                cte_add(lineno, line);
                                break;
                        }
                        if (tz == NULL || strcmp(&line[cursor], get_obj(tz))) {
                                rel_shared(tz);
                                tz = create_shared_str(&line[cursor]);
                        }
                        continue;
                }

                if (strncmp(&line[cursor], ENV_HOME,
                    strlen(ENV_HOME)) == 0) {
                        if ((tmp = strchr(&line[cursor], '\n')) != NULL) {
                                *tmp = NULL;
                        }
                        if (home == NULL ||
                            strcmp(&line[cursor], get_obj(home))) {
                                rel_shared(home);
                                home = create_shared_str(
                                    &line[cursor + strlen(ENV_HOME)]);
                        }
                        continue;
                }
```

```
1198                if (strncmp(&line[cursor], ENV_SHELL,
1199                    strlen(ENV_SHELL)) == 0) {
1200                        if ((tmp = strchr(&line[cursor], '\n')) != NULL) {
1201                                *tmp = NULL;
1202                        }
1203                        if (shell == NULL ||
1204                            strcmp(&line[cursor], get_obj(shell))) {
1205                                rel_shared(shell);
1206                                shell = create_shared_str(&line[cursor]);
1207                        }
1208                        continue;
1209                }

1211                e = xmalloc(sizeof (struct event));
1212                e->etype = CRONEVENT;
1213                if (!(((e->of.ct.minute = next_field(0, 59)) != NULL) &&
1214                    ((e->of.ct.hour = next_field(0, 23)) != NULL) &&
1215                    ((e->of.ct.daymon = next_field(1, 31)) != NULL) &&
1216                    ((e->of.ct.month = next_field(1, 12)) != NULL) &&
1217                    ((e->of.ct.dayweek = next_field(0, 6)) != NULL))) {
1218                        free(e);
1219                        cte_add(lineno, line);
1220                        continue;
1221                }
1222                while (line[cursor] == ' ' || line[cursor] == '\t')
1223                        cursor++;
1224                if (line[cursor] == '\n' || line[cursor] == '\0')
1225                        continue;
1226                /* get the command to execute   */
1227                start = cursor;
1228 again:
1229                while ((line[cursor] != '%') &&
1230                    (line[cursor] != '\n') &&
1231                    (line[cursor] != '\0') &&
1232                    (line[cursor] != '\\'))
1233                        cursor++;
1234                if (line[cursor] == '\\') {
1235                        cursor += 2;
1236                        goto again;
1237                }
1238                e->cmd = xmalloc(cursor-start + 1);
1239                (void) strncpy(e->cmd, line + start, cursor-start);
1240                e->cmd[cursor-start] = '\0';
1241                /* see if there is any standard input   */
1242                if (line[cursor] == '%') {
1243                        e->of.ct.input = xmalloc(strlen(line)-cursor + 1);
1244                        (void) strcpy(e->of.ct.input, line + cursor + 1);
1245                        for (i = 0; i < strlen(e->of.ct.input); i++) {
1246                                if (e->of.ct.input[i] == '%')
1247                                        e->of.ct.input[i] = '\n';
1248                        }
1249                } else {
1250                        e->of.ct.input = NULL;
1251                }
1252                /* set the timezone of this entry */
1253                e->of.ct.tz = dup_shared(tz);
1254                /* set the shell of this entry */
1255                e->of.ct.shell = dup_shared(shell);
1256                /* set the home of this entry */
1257                e->of.ct.home = dup_shared(home);
1258                /* have the event point to it's owner   */
1259                e->u = u;
1260                /* insert this event at the front of this user's event list */
1261                e->link = u->ctevents;
1262                u->ctevents = e;
1263                /* set the time for the first occurance of this event   */
```



```
1264                e->time = next_time(e, reftime);
1265                /* finally, add this event to the main event list       */
1266                switch (el_add(e, e->time, u->ctid)) {
1267                case -1:
1268                        ignore_msg("readcron", "cron", e);
1269                        break;
1270                case -2: /* event time lower than init time */
1271                        reset_needed = 1;
1272                        break;
1273                }
1274                cte_valid();
1275 #ifdef DEBUG
1276                cftime(timebuf, "%+", &e->time);
1274                cftime(timebuf, "%C", &e->time);
1277                (void) fprintf(stderr, "inserting cron event %s at %ld (%s)\n",
1278                    e->cmd, e->time, timebuf);
1279 #endif
1280        }
1281        cte_sendmail(u->name);  /* mail errors if any to user */
1282        (void) fclose(cf);
1283        rel_shared(tz);
1284        rel_shared(shell);
1285        rel_shared(home);
1286 }
______unchanged_portion_omitted_
```

```
**********************************************************
   21434 Wed Jan  1 08:36:45 2014
new/usr/src/cmd/fruadm/fruadm.c
4378 Clean up %C in *time() functions
438 need documentation for strftime %s flag
**********************************************************
```

```
/*
 * 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) 2014 Gary Mills
 *
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <libintl.h>
#include <libfru.h>
#include <errno.h>
#include <math.h>
#include <alloca.h>
#include <assert.h>
#include <sys/systeminfo.h>

#define NUM_OF_SEGMENT  1
#define SEGMENT_NAME_SIZE       2

#define FD_SEGMENT_SIZE 2949

static char  *command, *customer_data = NULL, *frupath = NULL, **svcargv;

/* DataElement supported in the customer operation */
static  char    *cust_data_list[] = {"Customer_DataR"};

/* DataElement supported in the service operation */
static  char    *serv_data_list[] = {"InstallationR", "ECO_CurrentR"};

/* currently supported segment name */
static  char    *segment_name[] = {"FD"};

static int   found_frupath = 0, list_only = 0, recursive = 0,
    service_mode = 0, svcargc, update = 0;
```



```
static void
usage(void)
{
        (void) fprintf(stderr,
            gettext("Usage:  %s [ -l ] | [ [ -r ] frupath [ text ] ]\n"),
            command);
}
```

*unchanged_portion_omitted_*

```
static void
displayBinary(unsigned char *data, size_t length, fru_elemdef_t *def)
{
        int     i = 0;
        uint64_t        lldata;
        uint64_t        mask;

        if (def->disp_type == FDISP_Hex) {
                for (i = 0; i < length; i++) {
                        (void) printf("%02X", data[i]);
                }
                return;
        }

        (void) memcpy(&lldata, data, sizeof (lldata));
        switch (def->disp_type) {
                case FDISP_Binary:
                {
                        mask = 0x8000000000000000ULL;
                        for (i = 0; i < (sizeof (uint64_t) *8); i++) {
                                if (lldata & (mask >> i)) {
                                        (void) printf("1");
                                } else {
                                        (void) printf("0");
                                }
                        }
                        return;
                }
                case FDISP_Octal:
                {
                        (void) printf("%llo", lldata);
                        return;
                }
                case FDISP_Decimal:
                {
                        (void) printf("%lld", lldata);
                        return;
                }
                case FDISP_Time:
                {
                        char buffer[PATH_MAX];
                        time_t time;
                        time = (time_t)lldata;
                        (void) strftime(buffer, PATH_MAX, "%+",
                        (void) strftime(buffer, PATH_MAX, "%C",
                            localtime(&time));
                        (void) printf("%s", buffer);
                        return;
                }
        }
}
```

*unchanged_portion_omitted_*

```
**********************************************************
    2683 Wed Jan  1 08:36:45 2014
new/usr/src/lib/libc/port/gen/cftime.c
4378 Clean up %C in *time() functions
438 need documentation for strftime %s flag
**********************************************************
/*
 * 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) 2014 Gary Mills
 *
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*      Copyright (c) 1988 AT&T */
/*        All Rights Reserved   */

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

/*
 * This routine converts time as follows.  The epoch is 0000  Jan  1
 * 1970  GMT.   The  argument  time  is  in seconds since then.  The
 * localtime(t) entry returns a pointer to an array containing:
 *
 *                seconds (0-59)
 *                minutes (0-59)
 *                hours (0-23)
 *                day of month (1-31)
 *                month (0-11)
 *                year
 *                weekday (0-6, Sun is 0)
 *                day of the year
 *                daylight savings flag
 *
 * The routine corrects for daylight saving time and  will  work  in
 * any  time  zone provided "timezone" is adjusted to the difference
 * between Greenwich and local standard time (measured in seconds).
 *
 *       ascftime(buf, format, t)       -> where t is produced by localtime
 *                                         and returns a ptr to a character
 *                                         string that has the ascii time in
 *                                         the format specified by the format
 *                                         argument (see date(1) for format
 *                                         syntax).
 *
 *       cftime(buf, format, t)         -> just calls ascftime.
```



```
 *
 *
 */

#include        "lint.h"
#include        <mtlib.h>
#include        <stddef.h>
#include        <time.h>
#include        <limits.h>
#include        <stdlib.h>
#include        <thread.h>
#include        <synch.h>

int
cftime(char *buf, char *format, const time_t *t)
{
        struct tm res;
        struct tm *p;

        p = localtime_r(t, &res);
        if (p == NULL) {
                *buf = '\0';
                return (0);
        }
        /* LINTED do not use ascftime() */
        return (ascftime(buf, format, p));
}

int
ascftime(char *buf, const char *format, const struct tm *tm)
{
        /* Set format string, if not already set */
        if (format == NULL || *format == '\0')
                if (((format = getenv("CFTIME")) == 0) || *format == 0)
                        format =  "%+";
                        format =  "%C";

        return ((int)strftime(buf, LONG_MAX, format, tm));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    9318 Wed Jan  1 08:36:46 2014
new/usr/src/lib/libfru/libnvfru/nvfru.c
4378 Clean up %C in *time() functions
438 need documentation for strftime %s flag
**********************************************************
```

```
/*
 * 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) 2014 Gary Mills
 *
 * Copyright (c) 2009, 2010, Oracle and/or its affiliates. All rights reserved.
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <strings.h>
#include <assert.h>
#include <pthread.h>
#include <sys/byteorder.h>
#include <sys/types.h>
#include <sys/nvpair.h>

#include "libfru.h"
#include "libfrup.h"
#include "fru_tag.h"
#include "libfrureg.h"
#include "nvfru.h"

#define NUM_ITER_BYTES  4
#define HEAD_ITER       0
#define TAIL_ITER       1
#define NUM_ITER        2
#define MAX_ITER        3
#define TIMESTRINGLEN   128

#define PARSE_TIME      1

static pthread_mutex_t gLock = PTHREAD_MUTEX_INITIALIZER;



static void
convert_field(const uint8_t *field, const fru_regdef_t *def, const char *path,
    nvlist_t *nv)
{
```



```
        char timestring[TIMESTRINGLEN];
        int i;
        uint64_t value;
        time_t timefield;

        switch (def->dataType) {
        case FDTYPE_Binary:
                assert(def->payloadLen <= sizeof (value));
                switch (def->dispType) {
#if PARSE_TIME == 1
                case FDISP_Time:
                        if (def->payloadLen > sizeof (timefield)) {
                                /* too big for formatting */
                                return;
                        }
                        (void) memcpy(&timefield, field, sizeof (timefield));
                        timefield = BE_32(timefield);
                        if (strftime(timestring, sizeof (timestring), "%c",
                        if (strftime(timestring, sizeof (timestring), "%C",
                            localtime(&timefield)) == 0) {
                                /* buffer too small */
                                return;
                        }
                        (void) nvlist_add_string(nv, path, timestring);
                        return;
#endif

                case FDISP_Binary:
                case FDISP_Octal:
                case FDISP_Decimal:
                case FDISP_Hex:
                default:
                        value = 0;
                        (void) memcpy((((uint8_t *)&value) +
                            sizeof (value) - def->payloadLen),
                            field, def->payloadLen);
                        value = BE_64(value);
                        switch (def->payloadLen) {
                        case 1:
                                (void) nvlist_add_uint8(nv, path,
                                    (uint8_t)value);
                                break;
                        case 2:
                                (void) nvlist_add_uint16(nv, path,
                                    (uint16_t)value);
                                break;
                        case 4:
                                (void) nvlist_add_uint32(nv, path,
                                    (uint32_t)value);
                                break;
                        default:
                                (void) nvlist_add_uint64(nv, path, value);
                        }
                        return;
                }

        case FDTYPE_ASCII:
                (void) nvlist_add_string(nv, path, (char *)field);
                return;

        case FDTYPE_Enumeration:
                value = 0;
                (void) memcpy((((uint8_t *)&value) + sizeof (value) -
                    def->payloadLen), field, def->payloadLen);
                value = BE_64(value);
                for (i = 0; i < def->enumCount; i++) {
```

```
126                         if (def->enumTable[i].value == value) {
127                                 (void) nvlist_add_string(nv, path,
128                                     def->enumTable[i].text);
129                                 return;
130                         }
131                 }
132         }

134         /* nothing matched above, use byte array */
135         (void) nvlist_add_byte_array(nv, path, (uchar_t *)field,
136             def->payloadLen);
137 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   25581 Wed Jan  1 08:36:46 2014
new/usr/src/lib/print/libpapi-common/common/attribute.c
4378 Clean up %C in *time() functions
438 need documentation for strftime %s flag
**********************************************************
```

```
/*
 * 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) 2014 Gary Mills
 *
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 */

/* $Id: attribute.c 157 2006-04-26 15:07:55Z ktou $ */

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

/*LINTLIBRARY*/

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <ctype.h>
#include <alloca.h>
#include <papi.h>
#include <regex.h>

#define MAX_PAGES 32767
/*
 * Assuming the maximum number of pages in
 * a document to be 32767
 */

static void papiAttributeFree(papi_attribute_t *attribute);

static void
papiAttributeValueFree(papi_attribute_value_type_t type,
                       papi_attribute_value_t *value)
{
        if (value != NULL) {
                switch (type) {
                case PAPI_STRING:
                        if (value->string != NULL)
```



```
                                free(value->string);
                        break;
                case PAPI_COLLECTION:
                        if (value->collection != NULL) {
                                int i;

                                for (i = 0; value->collection[i] != NULL; i++)
                                        papiAttributeFree(value->collection[i]);

                                free(value->collection);
                        }
                        break;
                default: /* don't need to free anything extra */
                        break;
                }

                free(value);
        }
}
```

*unchanged_portion_omitted_*

```
static papi_status_t
papiAttributeToString(papi_attribute_t *attribute, char *delim,
                char *buffer, size_t buflen)
{
        papi_attribute_value_t **values = attribute->values;
        int rc, i;

        if ((attribute->type == PAPI_BOOLEAN) && (values[1] == NULL)) {
                if (values[0]->boolean == PAPI_FALSE) {
                        if (isupper(attribute->name[0]) == 0)
                                strlcat(buffer, "no", buflen);
                        else
                                strlcat(buffer, "No", buflen);
                }
                rc = strlcat(buffer, attribute->name, buflen);
        } else {
                strlcat(buffer, attribute->name, buflen);
                rc = strlcat(buffer, "=", buflen);
        }

        if (values == NULL)
                return (PAPI_OK);

        for (i = 0; values[i] != NULL; i++) {
                switch (attribute->type) {
                case PAPI_STRING:
                        rc = strlcat(buffer, values[i]->string, buflen);
                        break;
                case PAPI_INTEGER: {
                        char string[24];

                        snprintf(string, sizeof (string), "%d",
                                values[i]->integer);
                        rc = strlcat(buffer, string, buflen);
                        }
                        break;
                case PAPI_BOOLEAN:
                        if (values[1] != NULL)
                                rc = strlcat(buffer, (values[i]->boolean ?
                                                "true" : "false"), buflen);
                        break;
                case PAPI_RANGE: {
                        char string[24];

                        if (values[i]->range.lower == values[i]->range.upper)
```

```
934                                 snprintf(string, sizeof (string), "%d",
935                                         values[i]->range.lower);
936                         else
937                                 snprintf(string, sizeof (string), "%d-%d",
938                                         values[i]->range.lower,
939                                         values[i]->range.upper);
940                         rc = strlcat(buffer, string, buflen);
941                         }
942                         break;
943                 case PAPI_RESOLUTION: {
944                         char string[24];

946                         snprintf(string, sizeof (string), "%dx%ddp%c",
947                                 values[i]->resolution.xres,
948                                 values[i]->resolution.yres,
949                                 (values[i]->resolution.units == PAPI_RES_PER_CM
950                                                         ? 'c' : 'i'));
951                         rc = strlcat(buffer, string, buflen);
952                         }
953                         break;
954                 case PAPI_DATETIME: {
955                         struct tm *tm = localtime(&values[i]->datetime);

957                         if (tm != NULL) {
958                                 char string[64];

960                                 strftime(string, sizeof (string), "%c", tm);
960                                 strftime(string, sizeof (string), "%C", tm);
961                                 rc = strlcat(buffer, string, buflen);
962                         }}
963                         break;
964                 case PAPI_COLLECTION: {
965                         char *string = alloca(buflen);

967                         papiAttributeListToString(values[i]->collection,
968                                         delim, string, buflen);
969                         rc = strlcat(buffer, string, buflen);
970                         }
971                         break;
972                 default: {
973                         char string[32];

975                         snprintf(string, sizeof (string), "unknown-type-0x%x",
976                                 attribute->type);
977                         rc = strlcat(buffer, string, buflen);
978                         }
979                 }
980                 if (values[i+1] != NULL)
981                         rc = strlcat(buffer, ",", buflen);

983                 if (rc >= buflen)
984                         return (PAPI_NOT_POSSIBLE);

986         }

988         return (PAPI_OK);
989 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
   15403 Wed Jan  1 08:36:46 2014
new/usr/src/man/man3c/strftime.3c
4378 Clean up %C in *time() functions
438 need documentation for strftime %s flag
**********************************************************
'\" te
.\" Copyright (c) 2014 Gary Mills
.\" Copyright (c) 2008, Sun Microsystems, Inc.  All Rights Reserved.
.\" Copyright 1989 AT&T
.\" Portions Copyright (c) 1992, X/Open Company Limited.  All Rights Reserved.
.\" Sun Microsystems, Inc. gratefully acknowledges The Open Group for permission
.\" http://www.opengroup.org/bookstore/.
.\" The Institute of Electrical and Electronics Engineers and The Open Group, ha
.\"  This notice shall appear on any product containing this material.
.\" The contents of this file are subject to the terms of the Common Development
.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
.\" When distributing Covered Code, include this CDDL HEADER in each file and in
.TH STRFTIME 3C "Jan 1, 2014"
.TH STRFTIME 3C "Sep 5, 2006"
.SH NAME
strftime, cftime, ascftime \- convert date and time to string
.SH SYNOPSIS
.LP
.nf
#include <time.h>

\fBsize_t\fR \fBstrftime\fR(\fBchar *restrict\fR \fIs\fR, \fBsize_t\fR \fImaxsiz
     \fBconst char *restrict\fR \fIformat\fR,
     \fBconst struct tm *restrict\fR \fItimeptr\fR);
.fi

.LP
.nf
\fBint\fR \fBcftime\fR(\fBchar *\fR\fIs\fR, \fBchar *\fR\fIformat\fR, \fBconst t
.fi

.LP
.nf
\fBint\fR \fBascftime\fR(\fBchar *\fR\fIs\fR, \fBconst char *\fR\fIformat\fR,
     \fBconst struct tm *\fR\fItimeptr\fR);
.fi

.SH DESCRIPTION
.sp
.LP
The \fBstrftime()\fR, \fBascftime()\fR, and \fBcftime()\fR functions place
bytes into the array pointed to by \fIs\fR as controlled by the string pointed
to by \fIformat\fR. The \fIformat\fR string consists of zero or more conversion
specifications and ordinary characters.  A conversion specification consists of
a '\fB%\fR' (percent) character and one or two terminating conversion
characters that determine the conversion specification's behavior.  All
ordinary characters (including the terminating null byte) are copied unchanged
into the array pointed to by \fIs\fR. If copying takes place between objects
that overlap, the behavior is undefined. For \fBstrftime()\fR, no more than
\fImaxsize\fR bytes are placed into the array.
.sp
.LP
If \fIformat\fR is \fB(char *)0\fR, then the locale's default format is used.
For \fBstrftime()\fR the default format is the same as \fB%c\fR; for
\fBcftime()\fR and \fBascftime()\fR the default format is the same as \fB%+\fR.
\fBcftime()\fR and \fBascftime()\fR the default format is the same as \fB%C\fR.
\fBcftime()\fR and \fBascftime()\fR first try to use the value of the
environment variable \fBCFTIME\fR, and if that is undefined or empty, the
default format is used.
.sp
```



```
.LP
Each conversion specification is replaced by appropriate characters as
described in the following list. The appropriate characters are determined by
the \fBLC_TIME\fR category of the program's locale and by the values contained
in the structure pointed to by \fItimeptr\fR for \fBstrftime()\fR and
\fBascftime()\fR, and by the time represented by \fIclock\fR for
\fBcftime()\fR.
.sp
.ne 2
.na
\fB\fB%%\fR\fR
.ad
.RS 6n
Same as \fB%\fR.
.RE

.sp
.ne 2
.na
\fB\fB%a\fR\fR
.ad
.RS 6n
Locale's abbreviated weekday name.
.RE

.sp
.ne 2
.na
\fB\fB%A\fR\fR
.ad
.RS 6n
Locale's full weekday name.
.RE

.sp
.ne 2
.na
\fB\fB%b\fR\fR
.ad
.RS 6n
Locale's abbreviated month name.
.RE

.sp
.ne 2
.na
\fB\fB%B\fR\fR
.ad
.RS 6n
Locale's full month name.
.RE

.SS "Default"
.sp
.ne 2
.na
\fB\fB%c\fR\fR
.ad
.RS 6n
Locale's appropriate date and time representation.
Locale's appropriate date and time represented as:
.sp
.in +2
.nf
%a %b %d %H:%M:%S %Y
.fi
```

```
.in -2

This is the default behavior as well as standard-conforming behavior for
standards first supported by releases prior to Solaris 2.4. See
\fBstandards\fR(5).
.RE

.SS "Standard conforming"
.sp
.ne 2
.na
\fB\fB%c\fR\fR
.ad
.RS 6n
Locale's appropriate date and time represented as:
.sp
.in +2
.nf
%a %b %e %H:%M:%S %Y
.fi
.in -2

This is standard-conforming behavior for standards first supported by Solaris
2.4 through Solaris 10.
.RE

.SS "Default"
.sp
.ne 2
.na
\fB\fB%C\fR\fR
.ad
.RS 6n
Locale's date and time representation as produced by \fBdate\fR(1).
.sp
This is the default behavior as well as standard-conforming behavior for
standards first supported by releases prior to Solaris 2.4.
.RE

.SS "Standard conforming"
.sp
.ne 2
.na
\fB\fB%C\fR\fR
.ad
.RS 6n
Century number (the year divided by 100 and truncated to an integer as a
decimal number [01,99]).
.sp
This is standard-conforming behavior for standards first supported by Solaris
2.4 through Solaris 10.
.RE

.sp
.ne 2
.na
\fB\fB%d\fR\fR
.ad
.RS 6n
Day of month [01,31].
.RE

.sp
.ne 2
.na
\fB\fB%D\fR\fR
```



```
.ad
.RS 6n
Date as \fB%m\fR/\fB%d\fR/\fB%y\fR.
.RE

.sp
.ne 2
.na
\fB\fB%e\fR\fR
.ad
.RS 6n
Day of month [1,31]; single  digits are preceded by a space.
.RE

.sp
.ne 2
.na
\fB\fB%F\fR\fR
.ad
.RS 6n
Equivalent to \fB%Y\fR-\fB%m\fR-\fB%d\fR (the ISO 8601:2000 standard date
format).
.RE

.sp
.ne 2
.na
\fB\fB%g\fR\fR
.ad
.RS 6n
Week-based year within century [00,99].
.RE

.sp
.ne 2
.na
\fB\fB%G\fR\fR
.ad
.RS 6n
Week-based year, including the century [0000,9999].
.RE

.sp
.ne 2
.na
\fB\fB%h\fR\fR
.ad
.RS 6n
Locale's abbreviated month name.
.RE

.sp
.ne 2
.na
\fB\fB%H\fR\fR
.ad
.RS 6n
Hour (24-hour clock) [00,23].
.RE

.sp
.ne 2
.na
\fB\fB%I\fR\fR
.ad
.RS 6n
```

```
209 Hour (12-hour clock) [01,12].
210 .RE

212 .sp
213 .ne 2
214 .na
215 \fB\fB%j\fR\fR
216 .ad
217 .RS 6n
218 Day number of year [001,366].
219 .RE

221 .sp
222 .ne 2
223 .na
224 \fB\fB%k\fR\fR
225 .ad
226 .RS 6n
227 Hour (24-hour clock) [0,23]; single digits are preceded by a space.
228 .RE

230 .sp
231 .ne 2
232 .na
233 \fB\fB%l\fR\fR
234 .ad
235 .RS 6n
236 Hour (12-hour clock) [1,12]; single digits are preceded by a space.
237 .RE

239 .sp
240 .ne 2
241 .na
242 \fB\fB%m\fR\fR
243 .ad
244 .RS 6n
245 Month number [01,12].
246 .RE

248 .sp
249 .ne 2
250 .na
251 \fB\fB%M\fR\fR
252 .ad
253 .RS 6n
254 Minute [00,59].
255 .RE

257 .sp
258 .ne 2
259 .na
260 \fB\fB%n\fR\fR
261 .ad
262 .RS 6n
263 Insert a NEWLINE.
264 .RE

266 .sp
267 .ne 2
268 .na
269 \fB\fB%p\fR\fR
270 .ad
271 .RS 6n
272 Locale’s equivalent of either a.m. or p.m.
273 .RE
```



```
275 .sp
276 .ne 2
277 .na
278 \fB\fB%r\fR\fR
279 .ad
280 .RS 6n
281 Appropriate time representation in 12-hour clock format with \fB%p\fR.
282 .RE

284 .sp
285 .ne 2
286 .na
287 \fB\fB%R\fR\fR
288 .ad
289 .RS 6n
290 Time as \fB%H\fR:\fB%M\fR.
291 .RE

293 .sp
294 .ne 2
295 .na
296 \fB\fB%s\fR\fR
297 .ad
298 .RS 6n
299 Seconds since 00:00:00 UTC, January 1, 1970.
300 .RE

302 .sp
303 .ne 2
304 .na
305 \fB\fB%S\fR\fR
306 .ad
307 .RS 6n
308 Seconds [00,60]; the range of values is [00,60] rather than [00,59] to allow
309 for the occasional leap second.
310 .RE

312 .sp
313 .ne 2
314 .na
315 \fB\fB%t\fR\fR
316 .ad
317 .RS 6n
318 Insert a TAB.
319 .RE

321 .sp
322 .ne 2
323 .na
324 \fB\fB%T\fR\fR
325 .ad
326 .RS 6n
327 Time as \fB%H\fR:\fB%M\fR:\fB%S\fR.
328 .RE

330 .sp
331 .ne 2
332 .na
333 \fB\fB%u\fR\fR
334 .ad
335 .RS 6n
336 Weekday as a decimal number [1,7], with 1 representing Monday. See \fBNOTES\fR
337 below.
338 .RE

340 .sp
```

```
.ne 2
.na
\fB\fB%U\fR\fR
.ad
.RS 6n
Week number of year as a decimal number [00,53], with Sunday as the first day
of week 1.
.RE

.sp
.ne 2
.na
\fB\fB%v\fR\fR
.ad
.RS 6n
Date as \fB%e\fR-\fB%b\fR-\fB%Y\fR.
.RE

.sp
.ne 2
.na
\fB\fB%V\fR\fR
.ad
.RS 6n
The ISO 8601 week number as a decimal number [01,53]. In the ISO 8601
week-based system, weeks begin on a Monday and week 1 of the year is the week
that includes both January 4th and the first Thursday of the year.  If the
first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of
the last week of the preceding year.  See \fBNOTES\fR below.
.RE

.sp
.ne 2
.na
\fB\fB%w\fR\fR
.ad
.RS 6n
Weekday as a decimal number [0,6], with 0 representing Sunday.
.RE

.sp
.ne 2
.na
\fB\fB%W\fR\fR
.ad
.RS 6n
Week number of year as a decimal number [00,53], with Monday as the first day
of week 1.
.RE

.sp
.ne 2
.na
\fB\fB%x\fR\fR
.ad
.RS 6n
Locale’s appropriate date representation.
.RE

.sp
.ne 2
.na
\fB\fB%X\fR\fR
.ad
.RS 6n
Locale’s appropriate time representation.
```



```
.RE

.sp
.ne 2
.na
\fB\fB%y\fR\fR
.ad
.RS 6n
Year within century [00,99].
.RE

.sp
.ne 2
.na
\fB\fB%Y\fR\fR
.ad
.RS 6n
Year, including the century (for example 1993).
.RE

.sp
.ne 2
.na
\fB\fB%z\fR\fR
.ad
.RS 6n
Replaced by offset from UTC in ISO 8601:2000 standard format (\fB+hhmm\fR or
\fB-hhmm\fR), or by no characters if no time zone is determinable. For example,
"-0430" means 4 hours 30 minutes behind UTC (west of Greenwich). If
\fBtm_isdst\fR is zero, the standard time offset is used.  If \fBtm_isdst\fR is
greater than zero, the daylight savings time offset if used. If \fBtm_isdst\fR
is negative, no characters are returned.
.RE

.sp
.ne 2
.na
\fB\fB%Z\fR\fR
.ad
.RS 6n
Time zone name or abbreviation, or no bytes if no time zone information exists.
.RE

.sp
.ne 2
.na
\fB\fB%+\fR\fR
.ad
.RS 6n
Locale’s date and time representation as produced by \fBdate\fR(1).
.RE

.sp
.LP
If a conversion specification does not correspond to any of the above or to any
of the modified conversion specifications listed below, the behavior is
undefined and \fB0\fR is returned.
.sp
.LP
The difference between \fB%U\fR and \fB%W\fR (and also between modified
conversion specifications \fB%OU\fR and \fB%OW\fR) lies in which day is counted
as the first of the week. Week number 1 is the first week in January starting
with a Sunday for \fB%U\fR or a Monday for \fB%W\fR. Week number 0 contains
those days before the first Sunday or Monday in January for \fB%U\fR and
\fB%W\fR, respectively.
.SS "Modified Conversion Specifications"
```

```
.sp
.LP
Some conversion specifications can be modified by the \fBE\fR and \fBO\fR
modifiers to indicate that an alternate format or specification should be used
rather than the one normally used by the unmodified conversion specification.
If the alternate format or specification does not exist in the current locale,
the behavior will be as if the unmodified specification were used.
.sp
.ne 2
.na
\fB\fB%Ec\fR\fR
.ad
.RS 7n
Locale’s alternate appropriate date and time representation.
.RE

.sp
.ne 2
.na
\fB\fB%EC\fR\fR
.ad
.RS 7n
Name of the base year (period) in the locale’s alternate representation.
.RE

.sp
.ne 2
.na
\fB\fB%Eg\fR\fR
.ad
.RS 7n
Offset from \fB%EC\fR of the week-based year in the locale’s alternative
representation.
.RE

.sp
.ne 2
.na
\fB\fB%EG\fR\fR
.ad
.RS 7n
Full alternative representation of the week-based year.
.RE

.sp
.ne 2
.na
\fB\fB%Ex\fR\fR
.ad
.RS 7n
Locale’s alternate date representation.
.RE

.sp
.ne 2
.na
\fB\fB%EX\fR\fR
.ad
.RS 7n
Locale’s alternate time representation.
.RE

.sp
.ne 2
.na
\fB\fB%Ey\fR\fR
```



```
.ad
.RS 7n
Offset from \fB%EC\fR (year only) in the locale’s alternate representation.
.RE

.sp
.ne 2
.na
\fB\fB%EY\fR\fR
.ad
.RS 7n
Full alternate year representation.
.RE

.sp
.ne 2
.na
\fB\fB%Od\fR\fR
.ad
.RS 7n
Day of the month using the locale’s alternate numeric symbols.
.RE

.sp
.ne 2
.na
\fB\fB%Oe\fR\fR
.ad
.RS 7n
Same as \fB%Od\fR.
.RE

.sp
.ne 2
.na
\fB\fB%Og\fR\fR
.ad
.RS 7n
Week-based year (offset from \fB%C\fR) in the locale’s alternate representation
and using the locale’s alternate numeric symbols.
.RE

.sp
.ne 2
.na
\fB\fB%OH\fR\fR
.ad
.RS 7n
Hour (24-hour clock) using the locale’s alternate numeric symbols.
.RE

.sp
.ne 2
.na
\fB\fB%OI\fR\fR
.ad
.RS 7n
Hour (12-hour clock) using the locale’s alternate numeric symbols.
.RE

.sp
.ne 2
.na
\fB\fB%Om\fR\fR
.ad
.RS 7n
```

```
Month using the locale’s alternate numeric symbols.
.RE

.sp
.ne 2
.na
\fB\fB%OM\fR\fR
.ad
.RS 7n
Minutes using the locale’s alternate numeric symbols.
.RE

.sp
.ne 2
.na
\fB\fB%OS\fR\fR
.ad
.RS 7n
Seconds using the locale’s alternate numeric symbols.
.RE

.sp
.ne 2
.na
\fB\fB%Ou\fR\fR
.ad
.RS 7n
Weekday as a number in the locale’s alternate numeric symbols.
.RE

.sp
.ne 2
.na
\fB\fB%OU\fR\fR
.ad
.RS 7n
Week number of the year (Sunday as the first day of the week) using the
locale’s alternate numeric symbols.
.RE

.sp
.ne 2
.na
\fB\fB%Ow\fR\fR
.ad
.RS 7n
Number of the weekday (Sunday=0) using the  locale’s alternate numeric symbols.
.RE

.sp
.ne 2
.na
\fB\fB%OW\fR\fR
.ad
.RS 7n
Week number of the year (Monday as the first day of the week) using the
locale’s alternate numeric symbols.
.RE

.sp
.ne 2
.na
\fB\fB%Oy\fR\fR
.ad
.RS 7n
Year (offset from \fB%C\fR) in the locale’s alternate representation and using
```



```
the locale’s alternate numeric symbols.
.RE

.SS "Selecting the Output Language"
.sp
.LP
By default, the output of \fBstrftime()\fR, \fBcftime()\fR, and
\fBascftime()\fR appear in U.S. English. The user can request that the output
of \fBstrftime()\fR, \fBcftime()\fR, or \fBascftime()\fR be in a specific
language by setting the \fBLC_TIME\fR category using \fBsetlocale()\fR.
.SS "Time Zone"
.sp
.LP
Local time zone information is used as though \fBtzset\fR(3C) were called.
.SH RETURN VALUES
.sp
.LP
The \fBstrftime()\fR, \fBcftime()\fR, and \fBascftime()\fR functions return the
number of characters placed into the array pointed to by \fIs\fR, not including
the terminating null character. If the total number of resulting characters
including the terminating null character is more than \fImaxsize\fR,
\fBstrftime()\fR returns \fB0\fR and the contents of the array are
indeterminate.
.SH EXAMPLES
.LP
\fBExample 1 \fRAn example of the \fBstrftime()\fR function.
.sp
.LP
The following example illustrates the use of \fBstrftime()\fR for the
\fBPOSIX\fR locale. It shows what the string in \fIstr\fR would look like if
the structure pointed to by \fItmptr\fR contains the values corresponding to
Thursday, August 28, 1986 at 12:44:36.

.sp
.in +2
.nf
\fBstrftime (str, strsize, "%A %b %d %j", tmptr)\fR
.fi
.in -2

.sp
.LP
This results in \fIstr\fR containing "Thursday Aug 28 240".

.SH ATTRIBUTES
.sp
.LP
See \fBattributes\fR(5) for descriptions of the following attributes:
.sp

.sp
.TS
box;
c | c
l | l .
ATTRIBUTE TYPE  ATTRIBUTE VALUE
_
CSI     Enabled
_
Interface Stability     Committed
_
MT-Level        MT-Safe
_
Standard        See below.
.TE
```

```
737 .sp
738 .LP
739 For \fBstrftime()\fR, see \fBstandards\fR(5).
740 .SH SEE ALSO
741 .sp
742 .LP
743 \fBdate\fR(1), \fBctime\fR(3C), \fBmktime\fR(3C), \fBsetlocale\fR(3C),
744 \fBstrptime\fR(3C), \fBtzset\fR(3C), \fBTIMEZONE\fR(4), \fBzoneinfo\fR(4),
745 \fBattributes\fR(5), \fBenviron\fR(5), \fBstandards\fR(5)
746 .SH NOTES
747 .sp
748 .LP
749 The conversion specification for \fB%V\fR was changed in the Solaris 7 release.
750 This change was based on the public review draft of the ISO C9x standard at
751 that time. Previously, the specification stated that if the week containing 1
752 January had fewer than four days in the new year, it became week 53 of the
753 previous year. The ISO C9x standard committee subsequently recognized that that
754 specification had been incorrect.
755 .sp
756 .LP
757 The conversion specifications for \fB%g\fR, \fB%G\fR, \fB%Eg\fR, \fB%EG\fR, and
758 \fB%Og\fR were added in the Solaris 7 release.  This change was based on the
759 public review draft of the ISO C9x standard at that time. These specifications
760 are evolving.  If the ISO C9x standard is finalized with a different
761 conclusion, these specifications will change to conform to the ISO C9x standard
762 decision.
763 .sp
764 .LP
765 The conversion specification for \fB%u\fR was changed in the Solaris 8 release.
766 This change was based on the XPG4 specification.
767 .sp
768 .LP
769 If using the \fB%Z\fR specifier and \fBzoneinfo\fR timezones and if the input
770 date is outside the range 20:45:52 UTC, December  13, 1901 to 03:14:07 UTC,
771 January 19, 2038, the timezone name may not be correct.
```