```
**********************************************************
   20233 Thu Jan 24 09:49:54 2019
new/usr/src/cmd/fmtmsg/main.c
10131 fmtmsg is bitwise, not streetwise
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2004 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

/*
 * Copyright (c) 2018, Joyent, Inc.
 */
#pragma ident   "%Z%%M% %I%     %E% SMI"


/*
 * fmtmsg.c
 *
 * Contains:
 *      fmtmsg          Command that writes a message in the standard
 *                      message format.  May in future make these
 *                      messages available for logging.
 */


/*
 * Header files used:
 *      <stdio.h>       C Standard I/O function definitions
 *      <string.h>      C string-handling definitions
 *      <errno.h>       UNIX error-code "errno" definitions
 *      <fmtmsg.h>      Standard Message definitions
 */

#include        <stdio.h>
#include        <string.h>
#include        <errno.h>
#include        <fmtmsg.h>


/*
```

```
 * Externals referenced:
 *      strtol          Function that converts char strings to "long"
 *      fmtmsg          Function that writes a message in standard format
 *      getenv          Function that extracts an environment variable's
 *                      value
 *      malloc          Allocate memory from the memory pool
 *      free            Frees allocated memory
 *      getopt          Function that extracts arguments from the command-
 *      optarg          Points to option's argument (from getopt())
 *      optind          Option's argument index (from getopt())
 *      opterr          FLAG, write error if invalid option (for getopt())
 *                      line.
 *      exit            Exits the command
 */

extern  long            strtol();
extern  int             fmtmsg();
extern  char           *getenv();
extern  void           *malloc();
extern  void            free();
extern  int             getopt();
extern  char           *optarg;
extern  int             optind;
extern  int             opterr;
extern  void            exit();
```

```
/*
 * Local definitions
 */

/*
 * Local constants
 */


/*
 * Boolean constants
 *      TRUE    Boolean value for "true" (any bits on)
 *      FALSE   Boolean value for "false" (all bits off)
 */

#ifndef FALSE
#define FALSE           (0)
#endif

#ifndef TRUE
#define TRUE            (1)
#endif


#define CLASS           (MM_PRINT|MM_SOFT|MM_NRECOV|MM_UTIL)
#define BIGUSAGE        "fmtmsg [-a action] [-c class] [-l label] [-s severity]


/*
 * Local data-type definitions
 */

/*
 * Structure used for tables containing keywords and integer values
 */

struct sev_info {
        char   *keyword;
        int     value;
};
______unchanged_portion_omitted_
```

```
/*
 * fmtmsg [-a action] [-c classification] [-l label] [-s severity] [-t tag]
 *        [-u subclass[,subclass[,...]]] [text]
 *
 * Function:
 *      Writes a message in the standard format.  Typically used by shell
 *      scripts to write error messages to the user.
 *
 * Arguments:
 *      text            String that is the text of the message
 *
 * Options:
 *   -a action          String that describes user action to take to
 *                      correct the situation
 *   -c classification  Keyword that identifies the type of the message
 *   -l label           String that identifies the source of the message
 *   -s severity        Keyword that identifies the severity of the message
 *   -t tag             String that identifies the message (use unclear)
 *   -u sub_classes     Comma-list of keywords that refines the type of
 *                      the message
 *
 * Environment Variables Used:
 *      MSGVERB         Defines the pieces of a message the user expects
 *                      to see.  It is a list of keywords separated by
 *                      colons (':').
 *      SEV_LEVEL       Defines a list of auxiliary severity keywords, values,
 *                      and print-strings.  It is a list of fields separated
 *                      by colons (':').  Each field consists of three
 *                      elements, keyword, value (in octal, hex, or decimal),
 *                      and print-string, separated by commas (',').
 *
 * Needs:
 *
 * Open Issues:
 */

int
main(int argc, char **argv)
{

        /* Local automatic data */

        long                    class;          /* Classification (built) */

        int                     severity;       /* User specified severity */
        int                     msgrtn;         /* Value returned by fmtmsg() */
        int                     optchar;        /* Opt char on cmdline */
        int                     exitval;        /* Value to return */

        int                     found;          /* FLAG, kywd found yet? */
        int                     errflg;         /* FLAG, error seen in cmd */
        int                     a_seen;         /* FLAG, -a option seen */
        int                     c_seen;         /* FLAG, -c option seen */
        int                     l_seen;         /* FLAG, -l option seen */
        int                     s_seen;         /* FLAG, -s option seen */
        int                     t_seen;         /* FLAG, -t option seen */
        int                     u_seen;         /* FLAG, -u option seen */
        int                     text_seen;      /* FLAG, text seen */

        char                   *text;           /* Ptr to user's text */
        char                   *label;          /* Ptr to user's label */
        char                   *tag;            /* Ptr to user's tag */
        char                   *action;         /* Ptr to user's action str */
        char                   *sstr;           /* Ptr to -s (severity) arg */
        char                   *ustr;           /* Ptr to -u (subclass) arg */
        char                   *cstr;           /* Ptr to -c (class) arg */
        char                    *sevstrval;      /* Ptr to SEV_LEVEL argument */
        char                    *sevval;         /* Ptr to temp SEV_LEVEL arg */
        char                    *tokenptr;       /* Ptr to current token */
        char                    *cmdname;        /* Ptr to base command name */
        char                    *p;              /* Multipurpose ptr */

        struct class_info       *class_info;     /* Ptr to class/subclass info st
        struct sev_info         *sev_info;       /* Ptr to severity info struct *
        struct sevstr           *penvsev;        /* Ptr to SEV_LEVEL values */



        /*
         * fmtmsg
         */


        /* Initializations */


        /* Extract the base command name from the command */
        if ((p = strrchr(argv[0], '/')) == (char *) NULL)
            cmdname = argv[0];
        else
            cmdname = p+1;

        /* Build the label for messages from "fmtmsg" */
        (void) snprintf(labelbuf, sizeof (labelbuf), "UX:%s", cmdname);


        /*
         * Extract arguments from the command line
         */

        /* Initializations */

        opterr = 0;                     /* Disable messages from getopt() */
        errflg = FALSE;                 /* No errors seen yet */

        a_seen = FALSE;                 /* No action (-a) text seen yet */
        c_seen = FALSE;                 /* No classification (-c) seen yet */
        l_seen = FALSE;                 /* No label (-l) seen yet */
        s_seen = FALSE;                 /* No severity (-s) seen yet */
        t_seen = FALSE;                 /* No tag (-t) seen yet */
        u_seen = FALSE;                 /* No subclass (-u) seen yet */
        text_seen = FALSE;              /* No text seen yet */


        /*
         * If only the command name was used, write out a usage string to
         * the standard output file.
         */

        if (argc == 1) {
            (void) fputs(BIGUSAGE, stderr);
            exit(0);
        }


        /* Parce command line */
        while (((optchar = getopt(argc, argv, "a:c:l:s:t:u:")) != EOF) &&
               !errflg) {

            switch(optchar) {

            case 'a':           /* -a actiontext */
```

```
                if (!a_seen) {
                    action = optarg;
                    a_seen = TRUE;
                } else errflg = TRUE;
                break;

            case 'c':           /* -c classification */
                if (!c_seen) {
                    cstr = optarg;
                    c_seen = TRUE;
                } else errflg = TRUE;
                break;

            case 'l':           /* -l label */
                if (!l_seen) {
                    label = optarg;
                    l_seen = TRUE;
                } else errflg = TRUE;
                break;

            case 's':           /* -s severity */
                if (!s_seen) {
                    sstr = optarg;
                    s_seen = TRUE;
                } else errflg = TRUE;
                break;

            case 't':           /* -t tag */
                if (!t_seen) {
                    tag = optarg;
                    t_seen = TRUE;
                } else errflg = TRUE;
                break;

            case 'u':           /* -u subclasslist */
                if (!u_seen) {
                    ustr = optarg;
                    u_seen = TRUE;
                } else errflg = TRUE;
                break;

            case '?':           /* -? or unknown option */
            default:
                errflg = TRUE;
                break;

            } /* esac */
        }


        /* Get the text */
        if (!errflg) {
            if (argc == (optind+1)) {
                text = argv[optind];
                text_seen = TRUE;
            }
            else if (argc != optind) {
                errflg = TRUE;
            }
        }


        /* Report syntax errors */
        if (errflg) {
            (void) fputs(BIGUSAGE, stderr);
            exit(1);
        }


        /*
         * Classification.
         */

        class = 0L;
        if (c_seen) {

            /* Search for keyword in list */
            for (class_info = &classes[0] ;
                 (class_info->keyword != (char *) NULL) &&
                 (strcmp(cstr, class_info->keyword)) ;
                 class_info++) ;

            /* If invalid (keyword unknown), write a message and exit */
            if (class_info->keyword == (char *) NULL) {
                (void) snprintf(msgbuf, sizeof (msgbuf),
                        "Invalid class: %s", cstr);
                (void) fmtmsg(CLASS, labelbuf, MM_ERROR, msgbuf,
                              MM_NULLACT, MM_NULLTAG);
                exit(1);
            }

            /* Save classification */
            class = class_info->value;

        }


        /*
         * Subclassification.
         */

        if (u_seen) {

            errflg = FALSE;
            p = strcpy(malloc((unsigned int) strlen(ustr)+1), ustr);
            if ((tokenptr = strtok(p, ",")) == (char *) NULL) errflg = TRUE;
            else do {

                /* Got a keyword.  Look for it in keyword list */
                for (class_info = subclasses ;
                     (class_info->keyword != (char *) NULL) &&
                     (strcmp(tokenptr, class_info->keyword) != 0) ;
                     class_info++) ;

                /* If found in list and no conflict, remember in class */
                if ((class_info->keyword != (char *) NULL) && ((class & class_in
                    class |= class_info->value;
                else
                    errflg = TRUE;

            } while (!errflg && ((tokenptr = strtok((char *) NULL, ",")) != (cha

            if (errflg) {
                (void) snprintf(msgbuf, sizeof (msgbuf),
                        "Invalid subclass: %s", ustr);
                (void) fmtmsg(CLASS, labelbuf, MM_ERROR, msgbuf,
                              MM_NULLACT, MM_NULLTAG);
                exit(1);
            }

        }
```

```
        if (!c_seen && !u_seen) class = MM_NULLMC;
        if (!c_seen & !u_seen) class = MM_NULLMC;



        /*
         * Severity.
         */

        if (s_seen) {

            /* If the severity is specified as a number, use that value */
            severity = strtol(sstr, &p, 10);
            if (*p || (strlen(sstr) == 0)) {

                /* Look for the standard severities */
                for (sev_info = severities ;
                     (sev_info->keyword != (char *) NULL) &&
                     (strcmp(sstr, sev_info->keyword)) ;
                     sev_info++) ;

                /*
                 * If the "severity" argument is one of the standard keywords,
                 * remember it for fmtmsg().  Otherwise, look at the SEV_LEVEL
                 * environment variable for severity extensions.
                 */

                /* If the keyword is one of the standard ones, save severity */
                if (sev_info->keyword != (char *) NULL) severity = sev_info->val

                else {

                    /*
                     * Severity keyword may be one of the extended set, if any.
                     */

                    /* Get the value of the SEV_LEVEL environment variable */
                    found = FALSE;
                    if ((sevstrval = getenv(SEV_LEVEL)) != (char *) NULL) {
                        sevval = (char *) malloc((unsigned int) strlen(sevstrval
                        penvsev = getauxsevs(strcpy(sevval, sevstrval));
                        if (penvsev != (struct sevstr *) NULL) do {
                            if (strcmp(penvsev->sevkywd, sstr) == 0) {
                                severity = penvsev->sevvalue;
                                found = TRUE;
                            }
                            else {
                                free(penvsev);
                                penvsev = getauxsevs((char *) NULL);
                            }
                        } while (!found && (penvsev != (struct sevstr *) NULL));

                        if (found) free(penvsev);
                        free(sevval);
                    }

                    if (!found) {
                        (void) snprintf(msgbuf, sizeof (msgbuf),
                                "Invalid severity: %s", sstr);
                        (void) fmtmsg(CLASS, labelbuf, MM_ERROR, msgbuf,
                                      MM_NULLACT, MM_NULLTAG);
                        exit(1);
                    }

                }  /* <severity> is not one of the standard severities */
            }  /* <severity> is not numeric */

        }  /* if (s_seen) */

        else severity = MM_NULLSEV;


        /*
         * Other options
         */

        if (!a_seen) action = MM_NULLACT;
        if (!l_seen) label = MM_NULLLBL;
        if (!t_seen) tag = MM_NULLTAG;
        if (!text_seen) text = MM_NULLTXT;


        /*
         * Write the message
         */

        msgrtn = fmtmsg(class, label, severity, text, action ,tag);


        /*
         * Return appropriate value to the shell (or wherever)
         */

        exitval = 0;
        if (msgrtn == MM_NOTOK) exitval = 32;
        else {
            if (msgrtn & MM_NOMSG) exitval += 2;
            if (msgrtn & MM_NOCON) exitval += 4;
        }

        return(exitval);
}
```
______***unchanged_portion_omitted_***