```
**********************************************************
  192762 Thu Jul 17 08:07:04 2014
new/usr/src/cmd/zonecfg/zonecfg.c
4956 zonecfg won't use a valid pager
**********************************************************
```

```
   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  */

  22 /*
  23  * Copyright (c) 2003, 2010, Oracle and/or its affiliates. All rights reserved.
  24  * Copyright 2014 Nexenta Systems, Inc.  All rights reserved.
  25  * Copyright 2014 Gary Mills
  26  */

  28 /*
  29  * zonecfg is a lex/yacc based command interpreter used to manage zone
  30  * configurations.  The lexer (see zonecfg_lex.l) builds up tokens, which
  31  * the grammar (see zonecfg_grammar.y) builds up into commands, some of
  32  * which takes resources and/or properties as arguments.  See the block
  33  * comments near the end of zonecfg_grammar.y for how the data structures
  34  * which keep track of these resources and properties are built up.
  35  *
  36  * The resource/property data structures are inserted into a command
  37  * structure (see zonecfg.h), which also keeps track of command names,
  38  * miscellaneous arguments, and function handlers.  The grammar selects
  39  * the appropriate function handler, each of which takes a pointer to a
  40  * command structure as its sole argument, and invokes it.  The grammar
  41  * itself is "entered" (a la the Matrix) by yyparse(), which is called
  42  * from read_input(), our main driving function.  That in turn is called
  43  * by one of do_interactive(), cmd_file() or one_command_at_a_time(), each
  44  * of which is called from main() depending on how the program was invoked.
  45  *
  46  * The rest of this module consists of the various function handlers and
  47  * their helper functions.  Some of these functions, particularly the
  48  * X_to_str() functions, which maps command, resource and property numbers
  49  * to strings, are used quite liberally, as doing so results in a better
  50  * program w/rt I18N, reducing the need for translation notes.
  51  */

  53 #include <sys/mntent.h>
  54 #include <sys/varargs.h>
  55 #include <sys/sysmacros.h>

  57 #include <errno.h>
  58 #include <fcntl.h>
  59 #include <strings.h>
  60 #include <unistd.h>
  61 #include <ctype.h>
```



```
  62 #include <stdlib.h>
  63 #include <assert.h>
  64 #include <sys/stat.h>
  65 #include <zone.h>
  66 #include <arpa/inet.h>
  67 #include <netdb.h>
  68 #include <locale.h>
  69 #include <libintl.h>
  70 #include <alloca.h>
  71 #include <signal.h>
  72 #include <wait.h>
  73 #include <libtecla.h>
  74 #include <libzfs.h>
  75 #include <sys/brand.h>
  76 #include <libbrand.h>
  77 #include <sys/systeminfo.h>
  78 #include <libdladm.h>
  79 #include <libinetutil.h>
  80 #include <pwd.h>
  81 #include <inet/ip.h>

  83 #include <libzonecfg.h>
  84 #include "zonecfg.h"

  86 #if !defined(TEXT_DOMAIN)               /* should be defined by cc -D */
  87 #define TEXT_DOMAIN     "SYS_TEST"      /* Use this only if it wasn't */
  88 #endif

  90 #define PAGER   "/usr/bin/more"
  91 #define EXEC_PREFIX     "exec "
  92 #define EXEC_LEN        (strlen(EXEC_PREFIX))

  94 struct help {
  95         uint_t  cmd_num;
  96         char    *cmd_name;
  97         uint_t  flags;
  98         char    *short_usage;
  99 };
______unchanged_portion_omitted_

 912 /*
 913  * Return the input filename appended to each component of the path
 914  * or the filename itself if it is absolute.
 915  * Parameters: path string, file name, output string.
 916  */
 917 /* Copied almost verbatim from libtnfctl/prb_findexec.c */
 918 static const char *
 919 exec_cat(const char *s1, const char *s2, char *si)
 920 {
 921         char               *s;
 922         /* Number of remaining characters in s */
 923         int                     cnt = PATH_MAX + 1;

 925         s = si;
 926         while (*s1 && *s1 != ':') { /* Copy first component of path to si */
 927                 if (cnt > 0) {
 928                         *s++ = *s1++;
 929                         cnt--;
 930                 } else {
 931                         s1++;
 932                 }
 933         }
 934         if (si != s && cnt > 0) { /* Add slash if s2 is not absolute */
 935                 *s++ = '/';
 936                 cnt--;
 937         }
```

```
        while (*s2 && cnt > 0) { /* Copy s2 to si */
                *s++ = *s2++;
                cnt--;
        }
        *s = '\0';  /* Terminate the output string */
        return (*s1 ? ++s1 : NULL);  /* Return next path component or NULL */
}

/* Determine that a name exists in PATH */
/* Copied with changes from libtnfctl/prb_findexec.c */
static int
path_find(const char *name)
{
        const char       *pathstr;
        char             fname[PATH_MAX + 2];
        const char       *cp;
        struct stat      stat_buf;

        if ((pathstr = getenv("PATH")) == NULL) {
                if (geteuid() == 0 || getuid() == 0)
                        pathstr = "/usr/sbin:/usr/bin";
                else
                        pathstr = "/usr/bin:";
        }
        cp = strchr(name, '/') ? (const char *) "" : pathstr;

        do {
                cp = exec_cat(cp, name, fname);
                if (stat(fname, &stat_buf) != -1) {
                        /* successful find of the file */
                        return (0);
                }
        } while (cp != NULL);

        return (-1);
}

static FILE *
pager_open(void) {
        FILE *newfp;
        char *pager, *space;

        pager = getenv("PAGER");
        if (pager == NULL || *pager == '\0')
                pager = PAGER;

        space = strchr(pager, ' ');
        if (space)
                *space = '\0';
        if (path_find(pager) == 0) {
                if (space)
                        *space = ' ';
                if ((newfp = popen(pager, "w")) == NULL)
                        zerr(gettext("PAGER open failed (%s)."),
                            strerror(errno));
                return (newfp);
        } else {
                zerr(gettext("PAGER %s does not exist (%s)."),
                    pager, strerror(errno));
        }
        return (NULL);
}

static void
pager_close(FILE *fp) {
        int status;
```



```
        status = pclose(fp);
        if (status == -1)
                zerr(gettext("PAGER close failed (%s)."),
                    strerror(errno));
}

/*
 * Called with verbose TRUE when help is explicitly requested, FALSE for
 * unexpected errors.
 */

void
usage(boolean_t verbose, uint_t flags)
{
        FILE *fp = verbose ? stdout : stderr;
        FILE *newfp;
        boolean_t need_to_close = B_FALSE;
        char *pager, *space;
        int i;
        struct stat statbuf;

        /* don't page error output */
        if (verbose && interactive_mode) {
                if ((newfp = pager_open()) != NULL) {
                if ((pager = getenv("PAGER")) == NULL)
                        pager = PAGER;

                space = strchr(pager, ' ');
                if (space)
                        *space = '\0';
                if (stat(pager, &statbuf) == 0) {
                        if (space)
                                *space = ' ';
                        if ((newfp = popen(pager, "w")) != NULL) {
                        need_to_close = B_TRUE;
                        fp = newfp;
                }
                } else {
                        zerr(gettext("PAGER %s does not exist (%s)."),
                            pager, strerror(errno));
        }
        }

        if (flags & HELP_META) {
                (void) fprintf(fp, gettext("More help is available for the "
                    "following:\n"));
                (void) fprintf(fp, "\n\tcommands ('%s commands')\n",
                    cmd_to_str(CMD_HELP));
                (void) fprintf(fp, "\tsyntax ('%s syntax')\n",
                    cmd_to_str(CMD_HELP));
                (void) fprintf(fp, "\tusage ('%s usage')\n\n",
                    cmd_to_str(CMD_HELP));
                (void) fprintf(fp, gettext("You may also obtain help on any "
                    "command by typing '%s <command-name>.'\n"),
                    cmd_to_str(CMD_HELP));
        }
        if (flags & HELP_RES_SCOPE) {
                switch (resource_scope) {
                case RT_FS:
                        (void) fprintf(fp, gettext("The '%s' resource scope is "
                            "used to configure a file-system.\n"),
                            rt_to_str(resource_scope));
                        (void) fprintf(fp, gettext("Valid commands:\n"));
                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
                            pt_to_str(PT_DIR), gettext("<path>"));
```

```
1054                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1055                            pt_to_str(PT_SPECIAL), gettext("<path>"));
1056                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1057                            pt_to_str(PT_RAW), gettext("<raw-device>"));
1058                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1059                            pt_to_str(PT_TYPE), gettext("<file-system type>"));
1060                        (void) fprintf(fp, "\t%s %s %s\n", cmd_to_str(CMD_ADD),
1061                            pt_to_str(PT_OPTIONS),
1062                            gettext("<file-system options>"));
1063                        (void) fprintf(fp, "\t%s %s %s\n",
1064                            cmd_to_str(CMD_REMOVE), pt_to_str(PT_OPTIONS),
1065                            gettext("<file-system options>"));
1066                        (void) fprintf(fp, gettext("Consult the file-system "
1067                            "specific manual page, such as mount_ufs(1M), "
1068                            "for\ndetails about file-system options.  Note "
1069                            "that any file-system options with an\nembedded "
1070                            "'=' character must be enclosed in double quotes, "
1071                            /*CSTYLED*/
1072                            "such as \"%s=5\".\n"), MNTOPT_RETRY);
1073                        break;
1074                case RT_NET:
1075                        (void) fprintf(fp, gettext("The '%s' resource scope is "
1076                            "used to configure a network interface.\n"),
1077                            rt_to_str(resource_scope));
1078                        (void) fprintf(fp, gettext("Valid commands:\n"));
1079                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1080                            pt_to_str(PT_ADDRESS), gettext("<IP-address>"));
1081                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1082                            pt_to_str(PT_ALLOWED_ADDRESS),
1083                            gettext("<IP-address>"));
1084                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1085                            pt_to_str(PT_PHYSICAL), gettext("<interface>"));
1086                        (void) fprintf(fp, gettext("See ifconfig(1M) for "
1087                            "details of the <interface> string.\n"));
1088                        (void) fprintf(fp, gettext("%s %s is valid "
1089                            "if the %s property is set to %s, otherwise it "
1090                            "must not be set.\n"),
1091                            cmd_to_str(CMD_SET), pt_to_str(PT_ADDRESS),
1092                            pt_to_str(PT_IPTYPE), gettext("shared"));
1093                        (void) fprintf(fp, gettext("%s %s is valid "
1094                            "if the %s property is set to %s, otherwise it "
1095                            "must not be set.\n"),
1096                            cmd_to_str(CMD_SET), pt_to_str(PT_ALLOWED_ADDRESS),
1097                            pt_to_str(PT_IPTYPE), gettext("exclusive"));
1098                        (void) fprintf(fp, gettext("\t%s %s=%s\n%s %s "
1099                            "is valid if the %s or %s property is set, "
1100                            "otherwise it must not be set\n"),
1101                            cmd_to_str(CMD_SET),
1102                            pt_to_str(PT_DEFROUTER), gettext("<IP-address>"),
1103                            cmd_to_str(CMD_SET), pt_to_str(PT_DEFROUTER),
1104                            gettext(pt_to_str(PT_ADDRESS)),
1105                            gettext(pt_to_str(PT_ALLOWED_ADDRESS)));
1106                        break;
1107                case RT_DEVICE:
1108                        (void) fprintf(fp, gettext("The '%s' resource scope is "
1109                            "used to configure a device node.\n"),
1110                            rt_to_str(resource_scope));
1111                        (void) fprintf(fp, gettext("Valid commands:\n"));
1112                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1113                            pt_to_str(PT_MATCH), gettext("<device-path>"));
1114                        break;
1115                case RT_RCTL:
1116                        (void) fprintf(fp, gettext("The '%s' resource scope is "
1117                            "used to configure a resource control.\n"),
1118                            rt_to_str(resource_scope));
1119                        (void) fprintf(fp, gettext("Valid commands:\n"));
```



```
1120                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1121                            pt_to_str(PT_NAME), gettext("<string>"));
1122                        (void) fprintf(fp, "\t%s %s (%s=%s,%s=%s,%s=%s)\n",
1123                            cmd_to_str(CMD_ADD), pt_to_str(PT_VALUE),
1124                            pt_to_str(PT_PRIV), gettext("<priv-value>"),
1125                            pt_to_str(PT_LIMIT), gettext("<number>"),
1126                            pt_to_str(PT_ACTION), gettext("<action-value>"));
1127                        (void) fprintf(fp, "\t%s %s (%s=%s,%s=%s,%s=%s)\n",
1128                            cmd_to_str(CMD_REMOVE), pt_to_str(PT_VALUE),
1129                            pt_to_str(PT_PRIV), gettext("<priv-value>"),
1130                            pt_to_str(PT_LIMIT), gettext("<number>"),
1131                            pt_to_str(PT_ACTION), gettext("<action-value>"));
1132                        (void) fprintf(fp, "%s\n\t%s := privileged\n"
1133                            "\t%s := none | deny\n", gettext("Where"),
1134                            gettext("<priv-value>"), gettext("<action-value>"));
1135                        break;
1136                case RT_ATTR:
1137                        (void) fprintf(fp, gettext("The '%s' resource scope is "
1138                            "used to configure a generic attribute.\n"),
1139                            rt_to_str(resource_scope));
1140                        (void) fprintf(fp, gettext("Valid commands:\n"));
1141                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1142                            pt_to_str(PT_NAME), gettext("<name>"));
1143                        (void) fprintf(fp, "\t%s %s=boolean\n",
1144                            cmd_to_str(CMD_SET), pt_to_str(PT_TYPE));
1145                        (void) fprintf(fp, "\t%s %s=true | false\n",
1146                            cmd_to_str(CMD_SET), pt_to_str(PT_VALUE));
1147                        (void) fprintf(fp, gettext("or\n"));
1148                        (void) fprintf(fp, "\t%s %s=int\n", cmd_to_str(CMD_SET),
1149                            pt_to_str(PT_TYPE));
1150                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1151                            pt_to_str(PT_VALUE), gettext("<integer>"));
1152                        (void) fprintf(fp, gettext("or\n"));
1153                        (void) fprintf(fp, "\t%s %s=string\n",
1154                            cmd_to_str(CMD_SET), pt_to_str(PT_TYPE));
1155                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1156                            pt_to_str(PT_VALUE), gettext("<string>"));
1157                        (void) fprintf(fp, gettext("or\n"));
1158                        (void) fprintf(fp, "\t%s %s=uint\n",
1159                            cmd_to_str(CMD_SET), pt_to_str(PT_TYPE));
1160                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1161                            pt_to_str(PT_VALUE), gettext("<unsigned integer>"));
1162                        break;
1163                case RT_DATASET:
1164                        (void) fprintf(fp, gettext("The '%s' resource scope is "
1165                            "used to export ZFS datasets.\n"),
1166                            rt_to_str(resource_scope));
1167                        (void) fprintf(fp, gettext("Valid commands:\n"));
1168                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1169                            pt_to_str(PT_NAME), gettext("<name>"));
1170                        break;
1171                case RT_DCPU:
1172                        (void) fprintf(fp, gettext("The '%s' resource scope "
1173                            "configures the 'pools' facility to dedicate\na "
1174                            "subset of the system's processors to this zone "
1175                            "while it is running.\n"),
1176                            rt_to_str(resource_scope));
1177                        (void) fprintf(fp, gettext("Valid commands:\n"));
1178                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1179                            pt_to_str(PT_NCPUS),
1180                            gettext("<unsigned integer | range>"));
1181                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1182                            pt_to_str(PT_IMPORTANCE),
1183                            gettext("<unsigned integer>"));
1184                        break;
1185                case RT_PCAP:
```

```
1186                        (void) fprintf(fp, gettext("The '%s' resource scope is "
1187                            "used to set an upper limit (a cap) on the\n"
1188                            "percentage of CPU that can be used by this zone.  "
1189                            "A '%s' value of 1\ncorresponds to one cpu.  The "
1190                            "value can be set higher than 1, up to the total\n"
1191                            "number of CPUs on the system.  The value can "
1192                            "also be less than 1,\nrepresenting a fraction of "
1193                            "a cpu.\n"),
1194                            rt_to_str(resource_scope), pt_to_str(PT_NCPUS));
1195                        (void) fprintf(fp, gettext("Valid commands:\n"));
1196                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1197                            pt_to_str(PT_NCPUS), gettext("<unsigned decimal>"));
1198                        break;
1199                case RT_MCAP:
1200                        (void) fprintf(fp, gettext("The '%s' resource scope is "
1201                            "used to set an upper limit (a cap) on the\n"
1202                            "amount of physical memory, swap space and locked "
1203                            "memory that can be used by\nthis zone.\n"),
1204                            rt_to_str(resource_scope));
1205                        (void) fprintf(fp, gettext("Valid commands:\n"));
1206                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1207                            pt_to_str(PT_PHYSICAL),
1208                            gettext("<qualified unsigned decimal>"));
1209                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1210                            pt_to_str(PT_SWAP),
1211                            gettext("<qualified unsigned decimal>"));
1212                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1213                            pt_to_str(PT_LOCKED),
1214                            gettext("<qualified unsigned decimal>"));
1215                        break;
1216                case RT_ADMIN:
1217                        (void) fprintf(fp, gettext("The '%s' resource scope is "
1218                            "used to delegate specific zone management\n"
1219                            "rights to users and roles. These rights are "
1220                            "only applicable to this zone.\n"),
1221                            rt_to_str(resource_scope));
1222                        (void) fprintf(fp, gettext("Valid commands:\n"));
1223                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1224                            pt_to_str(PT_USER),
1225                            gettext("<single user or role name>"));
1226                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1227                            pt_to_str(PT_AUTHS),
1228                            gettext("<comma separated list>"));
1229                        break;
1230                }
1231                (void) fprintf(fp, gettext("And from any resource scope, you "
1232                    "can:\n"));
1233                (void) fprintf(fp, "\t%s\t%s\n", cmd_to_str(CMD_END),
1234                    gettext("(to conclude this operation)"));
1235                (void) fprintf(fp, "\t%s\t%s\n", cmd_to_str(CMD_CANCEL),
1236                    gettext("(to cancel this operation)"));
1237                (void) fprintf(fp, "\t%s\t%s\n", cmd_to_str(CMD_EXIT),
1238                    gettext("(to exit the zonecfg utility)"));
1239        }
1240        if (flags & HELP_USAGE) {
1241                (void) fprintf(fp, "%s:\t%s %s\n", gettext("usage"),
1242                    execname, cmd_to_str(CMD_HELP));
1243                (void) fprintf(fp, "\t%s -z <zone>\t\t\t(%s)\n",
1244                    execname, gettext("interactive"));
1245                (void) fprintf(fp, "\t%s -z <zone> <command>\n", execname);
1246                (void) fprintf(fp, "\t%s -z <zone> -f <command-file>\n",
1247                    execname);
1248        }
1249        if (flags & HELP_SUBCMDS) {
1250                (void) fprintf(fp, "%s:\n\n", gettext("Commands"));
1251                for (i = 0; i <= CMD_MAX; i++) {
```



```
1252                        (void) fprintf(fp, "%s\n", helptab[i].short_usage);
1253                        if (verbose)
1254                                (void) fprintf(fp, "\t%s\n\n", long_help(i));
1255                }
1256        }
1257        if (flags & HELP_SYNTAX) {
1258                if (!verbose)
1259                        (void) fprintf(fp, "\n");
1260                (void) fprintf(fp, "<zone> := [A-Za-z0-9][A-Za-z0-9_.-]*\n");
1261                (void) fprintf(fp, gettext("\t(except the reserved words "
1262                    "'%s' and anything starting with '%s')\n"), "global",
1263                    "SUNW");
1264                (void) fprintf(fp,
1265                    gettext("\tName must be less than %d characters.\n"),
1266                    ZONENAME_MAX);
1267                if (verbose)
1268                        (void) fprintf(fp, "\n");
1269        }
1270        if (flags & HELP_NETADDR) {
1271                (void) fprintf(fp, gettext("\n<net-addr> :="));
1272                (void) fprintf(fp,
1273                    gettext("\t<IPv4-address>[/<IPv4-prefix-length>] |\n"));
1274                (void) fprintf(fp,
1275                    gettext("\t\t<IPv6-address>/<IPv6-prefix-length> |\n"));
1276                (void) fprintf(fp,
1277                    gettext("\t\t<hostname>[/<IPv4-prefix-length>]\n"));
1278                (void) fprintf(fp, gettext("See inet(3SOCKET) for IPv4 and "
1279                    "IPv6 address syntax.\n"));
1280                (void) fprintf(fp, gettext("<IPv4-prefix-length> := [0-32]\n"));
1281                (void) fprintf(fp,
1282                    gettext("<IPv6-prefix-length> := [0-128]\n"));
1283                (void) fprintf(fp,
1284                    gettext("<hostname> := [A-Za-z0-9][A-Za-z0-9-.]*\n"));
1285        }
1286        if (flags & HELP_RESOURCES) {
1287                (void) fprintf(fp, "<%s> := %s | %s | %s | %s | %s |\n\t"
1288                    "%s | %s | %s | %s | %s\n\n",
1289                    gettext("resource type"), rt_to_str(RT_FS),
1290                    rt_to_str(RT_NET), rt_to_str(RT_DEVICE),
1291                    rt_to_str(RT_RCTL), rt_to_str(RT_ATTR),
1292                    rt_to_str(RT_DATASET), rt_to_str(RT_DCPU),
1293                    rt_to_str(RT_PCAP), rt_to_str(RT_MCAP),
1294                    rt_to_str(RT_ADMIN));
1295        }
1296        if (flags & HELP_PROPS) {
1297                (void) fprintf(fp, gettext("For resource type ... there are "
1298                    "property types ...:\n"));
1299                (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1300                    pt_to_str(PT_ZONENAME));
1301                (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1302                    pt_to_str(PT_ZONEPATH));
1303                (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1304                    pt_to_str(PT_BRAND));
1305                (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1306                    pt_to_str(PT_AUTOBOOT));
1307                (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1308                    pt_to_str(PT_BOOTARGS));
1309                (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1310                    pt_to_str(PT_POOL));
1311                (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1312                    pt_to_str(PT_LIMITPRIV));
1313                (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1314                    pt_to_str(PT_SCHED));
1315                (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1316                    pt_to_str(PT_IPTYPE));
1317                (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
```

```
1318                    pt_to_str(PT_HOSTID));
1319                (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1320                    pt_to_str(PT_FS_ALLOWED));
1321                (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1322                    pt_to_str(PT_MAXLWPS));
1323                (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1324                    pt_to_str(PT_MAXPROCS));
1325                (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1326                    pt_to_str(PT_MAXSHMMEM));
1327                (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1328                    pt_to_str(PT_MAXSHMIDS));
1329                (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1330                    pt_to_str(PT_MAXMSGIDS));
1331                (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1332                    pt_to_str(PT_MAXSEMIDS));
1333                (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1334                    pt_to_str(PT_SHARES));
1335                (void) fprintf(fp, "\t%s\t\t%s, %s, %s, %s, %s\n",
1336                    rt_to_str(RT_FS), pt_to_str(PT_DIR),
1337                    pt_to_str(PT_SPECIAL), pt_to_str(PT_RAW),
1338                    pt_to_str(PT_TYPE), pt_to_str(PT_OPTIONS));
1339                (void) fprintf(fp, "\t%s\t\t%s, %s, %s|%s\n", rt_to_str(RT_NET),
1340                    pt_to_str(PT_ADDRESS), pt_to_str(PT_ALLOWED_ADDRESS),
1341                    pt_to_str(PT_PHYSICAL), pt_to_str(PT_DEFROUTER));
1342                (void) fprintf(fp, "\t%s\t\t%s\n", rt_to_str(RT_DEVICE),
1343                    pt_to_str(PT_MATCH));
1344                (void) fprintf(fp, "\t%s\t\t%s, %s\n", rt_to_str(RT_RCTL),
1345                    pt_to_str(PT_NAME), pt_to_str(PT_VALUE));
1346                (void) fprintf(fp, "\t%s\t\t%s, %s, %s\n", rt_to_str(RT_ATTR),
1347                    pt_to_str(PT_NAME), pt_to_str(PT_TYPE),
1348                    pt_to_str(PT_VALUE));
1349                (void) fprintf(fp, "\t%s\t\t%s\n", rt_to_str(RT_DATASET),
1350                    pt_to_str(PT_NAME));
1351                (void) fprintf(fp, "\t%s\t%s, %s\n", rt_to_str(RT_DCPU),
1352                    pt_to_str(PT_NCPUS), pt_to_str(PT_IMPORTANCE));
1353                (void) fprintf(fp, "\t%s\t%s\n", rt_to_str(RT_PCAP),
1354                    pt_to_str(PT_NCPUS));
1355                (void) fprintf(fp, "\t%s\t%s, %s, %s\n", rt_to_str(RT_MCAP),
1356                    pt_to_str(PT_PHYSICAL), pt_to_str(PT_SWAP),
1357                    pt_to_str(PT_LOCKED));
1358                (void) fprintf(fp, "\t%s\t\t%s, %s\n", rt_to_str(RT_ADMIN),
1359                    pt_to_str(PT_USER), pt_to_str(PT_AUTHS));
1360        }
1361        if (need_to_close)
1362                (void) pager_close(fp);
1277                (void) pclose(fp);
1363 }
```

*unchanged_portion_omitted_*

```
5426 void
5427 info_func(cmd_t *cmd)
5428 {
5429        FILE *fp = stdout;
5430        boolean_t need_to_close = B_FALSE;
5346        char *pager, *space;
5431        int type;
5432        int res1, res2;
5433        uint64_t swap_limit;
5434        uint64_t locked_limit;
5351        struct stat statbuf;

5436        assert(cmd != NULL);

5438        if (initialize(B_TRUE) != Z_OK)
5439                return;
```



```
5441        /* don't page error output */
5442        if (interactive_mode) {
5443                if ((fp = pager_open()) != NULL)
5360                if ((pager = getenv("PAGER")) == NULL)
5361                        pager = PAGER;
5362                space = strchr(pager, ' ');
5363                if (space)
5364                        *space = '\0';
5365                if (stat(pager, &statbuf) == 0) {
5366                        if (space)
5367                                *space = ' ';
5368                        if ((fp = popen(pager, "w")) != NULL)
5444                        need_to_close = B_TRUE;
5445                else
5446                        fp = stdout;
5372                } else {
5373                        zerr(gettext("PAGER %s does not exist (%s)."),
5374                            pager, strerror(errno));
5375                }

5448                setbuf(fp, NULL);
5449        }

5451        if (!global_scope) {
5452                switch (resource_scope) {
5453                case RT_FS:
5454                        output_fs(fp, &in_progress_fstab);
5455                        break;
5456                case RT_NET:
5457                        output_net(fp, &in_progress_nwiftab);
5458                        break;
5459                case RT_DEVICE:
5460                        output_dev(fp, &in_progress_devtab);
5461                        break;
5462                case RT_RCTL:
5463                        output_rctl(fp, &in_progress_rctltab);
5464                        break;
5465                case RT_ATTR:
5466                        output_attr(fp, &in_progress_attrtab);
5467                        break;
5468                case RT_DATASET:
5469                        output_ds(fp, &in_progress_dstab);
5470                        break;
5471                case RT_DCPU:
5472                        output_pset(fp, &in_progress_psettab);
5473                        break;
5474                case RT_PCAP:
5475                        output_pcap(fp);
5476                        break;
5477                case RT_MCAP:
5478                        res1 = zonecfg_get_aliased_rctl(handle, ALIAS_MAXSWAP,
5479                            &swap_limit);
5480                        res2 = zonecfg_get_aliased_rctl(handle,
5481                            ALIAS_MAXLOCKEDMEM, &locked_limit);
5482                        output_mcap(fp, &in_progress_mcaptab, res1, swap_limit,
5483                            res2, locked_limit);
5484                        break;
5485                case RT_ADMIN:
5486                        output_auth(fp, &in_progress_admintab);
5487                        break;
5488                }
5489                goto cleanup;
5490        }

5492        type = cmd->cmd_res_type;
```

```
5494         if (gz_invalid_rt_property(type)) {
5495                 zerr(gettext("%s is not a valid property for the global zone."),
5496                     rt_to_str(type));
5497                 goto cleanup;
5498         }

5500         if (gz_invalid_resource(type)) {
5501                 zerr(gettext("%s is not a valid resource for the global zone."),
5502                     rt_to_str(type));
5503                 goto cleanup;
5504         }

5506         switch (cmd->cmd_res_type) {
5507         case RT_UNKNOWN:
5508                 info_zonename(handle, fp);
5509                 if (!global_zone) {
5510                         info_zonepath(handle, fp);
5511                         info_brand(handle, fp);
5512                         info_autoboot(handle, fp);
5513                         info_bootargs(handle, fp);
5514                 }
5515                 info_pool(handle, fp);
5516                 if (!global_zone) {
5517                         info_limitpriv(handle, fp);
5518                         info_sched(handle, fp);
5519                         info_iptype(handle, fp);
5520                         info_hostid(handle, fp);
5521                         info_fs_allowed(handle, fp);
5522                 }
5523                 info_aliased_rctl(handle, fp, ALIAS_MAXLWPS);
5524                 info_aliased_rctl(handle, fp, ALIAS_MAXPROCS);
5525                 info_aliased_rctl(handle, fp, ALIAS_MAXSHMMEM);
5526                 info_aliased_rctl(handle, fp, ALIAS_MAXSHMIDS);
5527                 info_aliased_rctl(handle, fp, ALIAS_MAXMSGIDS);
5528                 info_aliased_rctl(handle, fp, ALIAS_MAXSEMIDS);
5529                 info_aliased_rctl(handle, fp, ALIAS_SHARES);
5530                 if (!global_zone) {
5531                         info_fs(handle, fp, cmd);
5532                         info_net(handle, fp, cmd);
5533                         info_dev(handle, fp, cmd);
5534                 }
5535                 info_pset(handle, fp);
5536                 info_pcap(fp);
5537                 info_mcap(handle, fp);
5538                 if (!global_zone) {
5539                         info_attr(handle, fp, cmd);
5540                         info_ds(handle, fp, cmd);
5541                         info_auth(handle, fp, cmd);
5542                 }
5543                 info_rctl(handle, fp, cmd);
5544                 break;
5545         case RT_ZONENAME:
5546                 info_zonename(handle, fp);
5547                 break;
5548         case RT_ZONEPATH:
5549                 info_zonepath(handle, fp);
5550                 break;
5551         case RT_BRAND:
5552                 info_brand(handle, fp);
5553                 break;
5554         case RT_AUTOBOOT:
5555                 info_autoboot(handle, fp);
5556                 break;
5557         case RT_POOL:
5558                 info_pool(handle, fp);
5559                 break;
```



```
5560         case RT_LIMITPRIV:
5561                 info_limitpriv(handle, fp);
5562                 break;
5563         case RT_BOOTARGS:
5564                 info_bootargs(handle, fp);
5565                 break;
5566         case RT_SCHED:
5567                 info_sched(handle, fp);
5568                 break;
5569         case RT_IPTYPE:
5570                 info_iptype(handle, fp);
5571                 break;
5572         case RT_MAXLWPS:
5573                 info_aliased_rctl(handle, fp, ALIAS_MAXLWPS);
5574                 break;
5575         case RT_MAXPROCS:
5576                 info_aliased_rctl(handle, fp, ALIAS_MAXPROCS);
5577                 break;
5578         case RT_MAXSHMMEM:
5579                 info_aliased_rctl(handle, fp, ALIAS_MAXSHMMEM);
5580                 break;
5581         case RT_MAXSHMIDS:
5582                 info_aliased_rctl(handle, fp, ALIAS_MAXSHMIDS);
5583                 break;
5584         case RT_MAXMSGIDS:
5585                 info_aliased_rctl(handle, fp, ALIAS_MAXMSGIDS);
5586                 break;
5587         case RT_MAXSEMIDS:
5588                 info_aliased_rctl(handle, fp, ALIAS_MAXSEMIDS);
5589                 break;
5590         case RT_SHARES:
5591                 info_aliased_rctl(handle, fp, ALIAS_SHARES);
5592                 break;
5593         case RT_FS:
5594                 info_fs(handle, fp, cmd);
5595                 break;
5596         case RT_NET:
5597                 info_net(handle, fp, cmd);
5598                 break;
5599         case RT_DEVICE:
5600                 info_dev(handle, fp, cmd);
5601                 break;
5602         case RT_RCTL:
5603                 info_rctl(handle, fp, cmd);
5604                 break;
5605         case RT_ATTR:
5606                 info_attr(handle, fp, cmd);
5607                 break;
5608         case RT_DATASET:
5609                 info_ds(handle, fp, cmd);
5610                 break;
5611         case RT_DCPU:
5612                 info_pset(handle, fp);
5613                 break;
5614         case RT_PCAP:
5615                 info_pcap(fp);
5616                 break;
5617         case RT_MCAP:
5618                 info_mcap(handle, fp);
5619                 break;
5620         case RT_HOSTID:
5621                 info_hostid(handle, fp);
5622                 break;
5623         case RT_ADMIN:
5624                 info_auth(handle, fp, cmd);
5625                 break;
```

```
5626         case RT_FS_ALLOWED:
5627                 info_fs_allowed(handle, fp);
5628                 break;
5629         default:
5630                 zone_perror(rt_to_str(cmd->cmd_res_type), Z_NO_RESOURCE_TYPE,
5631                     B_TRUE);
5632         }

5634 cleanup:
5635         if (need_to_close)
5636                 (void) pager_close(fp);
5565                 (void) pclose(fp);
5637 }
```

_____***unchanged_portion_omitted_***