```
**********************************************************
  192048 Mon Jul  7 08:39:36 2014
new/usr/src/cmd/zonecfg/zonecfg.c
3347 zonecfg(1M) is confused about selection
4956 zonecfg won't use a valid pager
**********************************************************
```

```
/*
 * 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) 2003, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2014 Nexenta Systems, Inc.  All rights reserved.
 * Copyright 2014 Gary Mills
 */

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

#include <sys/mntent.h>
#include <sys/varargs.h>
#include <sys/sysmacros.h>

#include <errno.h>
#include <fcntl.h>
#include <strings.h>
#include <unistd.h>
```



```
#include <ctype.h>
#include <stdlib.h>
#include <assert.h>
#include <sys/stat.h>
#include <zone.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <locale.h>
#include <libintl.h>
#include <alloca.h>
#include <signal.h>
#include <wait.h>
#include <libtecla.h>
#include <libzfs.h>
#include <sys/brand.h>
#include <libbrand.h>
#include <sys/systeminfo.h>
#include <libdladm.h>
#include <libinetutil.h>
#include <pwd.h>
#include <inet/ip.h>

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

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

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

struct help {
        uint_t  cmd_num;
        char    *cmd_name;
        uint_t  flags;
        char    *short_usage;
};
```

*unchanged_portion_omitted_*

```
/*
 * 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 ((pager = getenv("PAGER")) == NULL)
                        pager = PAGER;

                space = strchr(pager, ' ');
                if (space)
                        *space = '\0';
                if (*pager == '/' && stat(pager, &statbuf) != 0) {
                        zerr(gettext("PAGER %s does not exist (%s)."),
```

```
                            pager, strerror(errno));
                } else {
                if (stat(pager, &statbuf) == 0) {
                        if (space)
                                *space = ' ';
                        if ((newfp = popen(pager, "w")) == NULL) {
                                zerr(gettext("PAGER %s open failed (%s)."),
                                    pager, strerror(errno));
                        } else {
                        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>"));
                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
                            pt_to_str(PT_SPECIAL), gettext("<path>"));
                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
                            pt_to_str(PT_RAW), gettext("<raw-device>"));
                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
                            pt_to_str(PT_TYPE), gettext("<file-system type>"));
                        (void) fprintf(fp, "\t%s %s %s\n", cmd_to_str(CMD_ADD),
                            pt_to_str(PT_OPTIONS),
                            gettext("<file-system options>"));
                        (void) fprintf(fp, "\t%s %s %s\n",
                            cmd_to_str(CMD_REMOVE), pt_to_str(PT_OPTIONS),
                            gettext("<file-system options>"));
                        (void) fprintf(fp, gettext("Consult the file-system "
                            "specific manual page, such as mount_ufs(1M), "
                            "for\ndetails about file-system options.  Note "
                            "that any file-system options with an\nembedded "
                            "'=' character must be enclosed in double quotes, "
                            /*CSTYLED*/
                            "such as \"%s=5\".\n"), MNTOPT_RETRY);
                        break;
                case RT_NET:
                        (void) fprintf(fp, gettext("The '%s' resource scope is "
                            "used to configure a network interface.\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_ADDRESS), gettext("<IP-address>"));
                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
                            pt_to_str(PT_ALLOWED_ADDRESS),
                            gettext("<IP-address>"));
                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
                            pt_to_str(PT_PHYSICAL), gettext("<interface>"));
                        (void) fprintf(fp, gettext("See ifconfig(1M) for "
                            "details of the <interface> string.\n"));
                        (void) fprintf(fp, gettext("%s %s is valid "
                            "if the %s property is set to %s, otherwise it "
                            "must not be set.\n"),
                            cmd_to_str(CMD_SET), pt_to_str(PT_ADDRESS),
                            pt_to_str(PT_IPTYPE), gettext("shared"));
                        (void) fprintf(fp, gettext("%s %s is valid "
                            "if the %s property is set to %s, otherwise it "
                            "must not be set.\n"),
                            cmd_to_str(CMD_SET), pt_to_str(PT_ALLOWED_ADDRESS),
                            pt_to_str(PT_IPTYPE), gettext("exclusive"));
                        (void) fprintf(fp, gettext("\t%s %s=%s\n%s %s "
                            "is valid if the %s or %s property is set, "
                            "otherwise it must not be set\n"),
                            cmd_to_str(CMD_SET),
                            pt_to_str(PT_DEFROUTER), gettext("<IP-address>"),
                            cmd_to_str(CMD_SET), pt_to_str(PT_DEFROUTER),
                            gettext(pt_to_str(PT_ADDRESS)),
                            gettext(pt_to_str(PT_ALLOWED_ADDRESS)));
                        break;
                case RT_DEVICE:
                        (void) fprintf(fp, gettext("The '%s' resource scope is "
                            "used to configure a device node.\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_MATCH), gettext("<device-path>"));
                        break;
                case RT_RCTL:
                        (void) fprintf(fp, gettext("The '%s' resource scope is "
                            "used to configure a resource control.\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_NAME), gettext("<string>"));
                        (void) fprintf(fp, "\t%s %s (%s=%s,%s=%s,%s=%s)\n",
                            cmd_to_str(CMD_ADD), pt_to_str(PT_VALUE),
                            pt_to_str(PT_PRIV), gettext("<priv-value>"),
                            pt_to_str(PT_LIMIT), gettext("<number>"),
                            pt_to_str(PT_ACTION), gettext("<action-value>"));
                        (void) fprintf(fp, "\t%s %s (%s=%s,%s=%s,%s=%s)\n",
                            cmd_to_str(CMD_REMOVE), pt_to_str(PT_VALUE),
                            pt_to_str(PT_PRIV), gettext("<priv-value>"),
                            pt_to_str(PT_LIMIT), gettext("<number>"),
                            pt_to_str(PT_ACTION), gettext("<action-value>"));
                        (void) fprintf(fp, "%s\n\t%s := privileged\n"
                            "\t%s := none | deny\n", gettext("Where"),
                            gettext("<priv-value>"), gettext("<action-value>"));
                        break;
                case RT_ATTR:
                        (void) fprintf(fp, gettext("The '%s' resource scope is "
                            "used to configure a generic attribute.\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_NAME), gettext("<name>"));
                        (void) fprintf(fp, "\t%s %s=boolean\n",
                            cmd_to_str(CMD_SET), pt_to_str(PT_TYPE));
```

```
1064                        (void) fprintf(fp, "\t%s %s=true | false\n",
1065                            cmd_to_str(CMD_SET), pt_to_str(PT_VALUE));
1066                        (void) fprintf(fp, gettext("or\n"));
1067                        (void) fprintf(fp, "\t%s %s=int\n", cmd_to_str(CMD_SET),
1068                            pt_to_str(PT_TYPE));
1069                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1070                            pt_to_str(PT_VALUE), gettext("<integer>"));
1071                        (void) fprintf(fp, gettext("or\n"));
1072                        (void) fprintf(fp, "\t%s %s=string\n",
1073                            cmd_to_str(CMD_SET), pt_to_str(PT_TYPE));
1074                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1075                            pt_to_str(PT_VALUE), gettext("<string>"));
1076                        (void) fprintf(fp, gettext("or\n"));
1077                        (void) fprintf(fp, "\t%s %s=uint\n",
1078                            cmd_to_str(CMD_SET), pt_to_str(PT_TYPE));
1079                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1080                            pt_to_str(PT_VALUE), gettext("<unsigned integer>"));
1081                        break;
1082                case RT_DATASET:
1083                        (void) fprintf(fp, gettext("The '%s' resource scope is "
1084                            "used to export ZFS datasets.\n"),
1085                            rt_to_str(resource_scope));
1086                        (void) fprintf(fp, gettext("Valid commands:\n"));
1087                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1088                            pt_to_str(PT_NAME), gettext("<name>"));
1089                        break;
1090                case RT_DCPU:
1091                        (void) fprintf(fp, gettext("The '%s' resource scope "
1092                            "configures the 'pools' facility to dedicate\na "
1093                            "subset of the system's processors to this zone "
1094                            "while it is running.\n"),
1095                            rt_to_str(resource_scope));
1096                        (void) fprintf(fp, gettext("Valid commands:\n"));
1097                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1098                            pt_to_str(PT_NCPUS),
1099                            gettext("<unsigned integer | range>"));
1100                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1101                            pt_to_str(PT_IMPORTANCE),
1102                            gettext("<unsigned integer>"));
1103                        break;
1104                case RT_PCAP:
1105                        (void) fprintf(fp, gettext("The '%s' resource scope is "
1106                            "used to set an upper limit (a cap) on the\n"
1107                            "percentage of CPU that can be used by this zone.  "
1108                            "A '%s' value of 1\ncorresponds to one cpu.  The "
1109                            "value can be set higher than 1, up to the total\n"
1110                            "number of CPUs on the system.  The value can "
1111                            "also be less than 1,\nrepresenting a fraction of "
1112                            "a cpu.\n"),
1113                            rt_to_str(resource_scope), pt_to_str(PT_NCPUS));
1114                        (void) fprintf(fp, gettext("Valid commands:\n"));
1115                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1116                            pt_to_str(PT_NCPUS), gettext("<unsigned decimal>"));
1117                        break;
1118                case RT_MCAP:
1119                        (void) fprintf(fp, gettext("The '%s' resource scope is "
1120                            "used to set an upper limit (a cap) on the\n"
1121                            "amount of physical memory, swap space and locked "
1122                            "memory that can be used by\nthis zone.\n"),
1123                            rt_to_str(resource_scope));
1124                        (void) fprintf(fp, gettext("Valid commands:\n"));
1125                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1126                            pt_to_str(PT_PHYSICAL),
1127                            gettext("<qualified unsigned decimal>"));
1128                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1129                            pt_to_str(PT_SWAP),
```



```
1130                            gettext("<qualified unsigned decimal>"));
1131                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1132                            pt_to_str(PT_LOCKED),
1133                            gettext("<qualified unsigned decimal>"));
1134                        break;
1135                case RT_ADMIN:
1136                        (void) fprintf(fp, gettext("The '%s' resource scope is "
1137                            "used to delegate specific zone management\n"
1138                            "rights to users and roles. These rights are "
1139                            "only applicable to this zone.\n"),
1140                            rt_to_str(resource_scope));
1141                        (void) fprintf(fp, gettext("Valid commands:\n"));
1142                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1143                            pt_to_str(PT_USER),
1144                            gettext("<single user or role name>"));
1145                        (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1146                            pt_to_str(PT_AUTHS),
1147                            gettext("<comma separated list>"));
1148                        break;
1149                }
1150                (void) fprintf(fp, gettext("And from any resource scope, you "
1151                    "can:\n"));
1152                (void) fprintf(fp, "\t%s\t%s\n", cmd_to_str(CMD_END),
1153                    gettext("(to conclude this operation)"));
1154                (void) fprintf(fp, "\t%s\t%s\n", cmd_to_str(CMD_CANCEL),
1155                    gettext("(to cancel this operation)"));
1156                (void) fprintf(fp, "\t%s\t%s\n", cmd_to_str(CMD_EXIT),
1157                    gettext("(to exit the zonecfg utility)"));
1158        }
1159        if (flags & HELP_USAGE) {
1160                (void) fprintf(fp, "%s:\t%s %s\n", gettext("usage"),
1161                    execname, cmd_to_str(CMD_HELP));
1162                (void) fprintf(fp, "\t%s -z <zone>\t\t\t(%s)\n",
1163                    execname, gettext("interactive"));
1164                (void) fprintf(fp, "\t%s -z <zone> <command>\n", execname);
1165                (void) fprintf(fp, "\t%s -z <zone> -f <command-file>\n",
1166                    execname);
1167        }
1168        if (flags & HELP_SUBCMDS) {
1169                (void) fprintf(fp, "%s:\n\n", gettext("Commands"));
1170                for (i = 0; i <= CMD_MAX; i++) {
1171                        (void) fprintf(fp, "%s\n", helptab[i].short_usage);
1172                        if (verbose)
1173                                (void) fprintf(fp, "\t%s\n\n", long_help(i));
1174                }
1175        }
1176        if (flags & HELP_SYNTAX) {
1177                if (!verbose)
1178                        (void) fprintf(fp, "\n");
1179                (void) fprintf(fp, "<zone> := [A-Za-z0-9][A-Za-z0-9_.-]*\n");
1180                (void) fprintf(fp, gettext("\t(except the reserved words "
1181                    "'%s' and anything starting with '%s')\n"), "global",
1182                    "SUNW");
1183                (void) fprintf(fp,
1184                    gettext("\tName must be less than %d characters.\n"),
1185                    ZONENAME_MAX);
1186                if (verbose)
1187                        (void) fprintf(fp, "\n");
1188        }
1189        if (flags & HELP_NETADDR) {
1190                (void) fprintf(fp, gettext("\n<net-addr> :="));
1191                (void) fprintf(fp,
1192                    gettext("\t<IPv4-address>[/<IPv4-prefix-length>] |\n"));
1193                (void) fprintf(fp,
1194                    gettext("\t\t<IPv6-address>/<IPv6-prefix-length> |\n"));
1195                (void) fprintf(fp,
```

```
1196                    gettext("\t\t<hostname>[/<IPv4-prefix-length>]\n"));
1197            (void) fprintf(fp, gettext("See inet(3SOCKET) for IPv4 and "
1198                "IPv6 address syntax.\n"));
1199            (void) fprintf(fp, gettext("<IPv4-prefix-length> := [0-32]\n"));
1200            (void) fprintf(fp,
1201                gettext("<IPv6-prefix-length> := [0-128]\n"));
1202            (void) fprintf(fp,
1203                gettext("<hostname> := [A-Za-z0-9][A-Za-z0-9-.]*\n"));
1204        }
1205        if (flags & HELP_RESOURCES) {
1206            (void) fprintf(fp, "<%s> := %s | %s | %s | %s | %s |\n\t"
1207                "%s | %s | %s | %s | %s\n\n",
1208                gettext("resource type"), rt_to_str(RT_FS),
1209                rt_to_str(RT_NET), rt_to_str(RT_DEVICE),
1210                rt_to_str(RT_RCTL), rt_to_str(RT_ATTR),
1211                rt_to_str(RT_DATASET), rt_to_str(RT_DCPU),
1212                rt_to_str(RT_PCAP), rt_to_str(RT_MCAP),
1213                rt_to_str(RT_ADMIN));
1214        }
1215        if (flags & HELP_PROPS) {
1216            (void) fprintf(fp, gettext("For resource type ... there are "
1217                "property types ...:\n"));
1218            (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1219                pt_to_str(PT_ZONENAME));
1220            (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1221                pt_to_str(PT_ZONEPATH));
1222            (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1223                pt_to_str(PT_BRAND));
1224            (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1225                pt_to_str(PT_AUTOBOOT));
1226            (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1227                pt_to_str(PT_BOOTARGS));
1228            (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1229                pt_to_str(PT_POOL));
1230            (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1231                pt_to_str(PT_LIMITPRIV));
1232            (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1233                pt_to_str(PT_SCHED));
1234            (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1235                pt_to_str(PT_IPTYPE));
1236            (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1237                pt_to_str(PT_HOSTID));
1238            (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1239                pt_to_str(PT_FS_ALLOWED));
1240            (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1241                pt_to_str(PT_MAXLWPS));
1242            (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1243                pt_to_str(PT_MAXPROCS));
1244            (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1245                pt_to_str(PT_MAXSHMMEM));
1246            (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1247                pt_to_str(PT_MAXSHMIDS));
1248            (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1249                pt_to_str(PT_MAXMSGIDS));
1250            (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1251                pt_to_str(PT_MAXSEMIDS));
1252            (void) fprintf(fp, "\t%s\t%s\n", gettext("(global)"),
1253                pt_to_str(PT_SHARES));
1254            (void) fprintf(fp, "\t%s\t\t%s, %s, %s, %s, %s\n",
1255                rt_to_str(RT_FS), pt_to_str(PT_DIR),
1256                pt_to_str(PT_SPECIAL), pt_to_str(PT_RAW),
1257                pt_to_str(PT_TYPE), pt_to_str(PT_OPTIONS));
1258            (void) fprintf(fp, "\t%s\t\t%s, %s, %s|%s\n", rt_to_str(RT_NET),
1259                pt_to_str(PT_ADDRESS), pt_to_str(PT_ALLOWED_ADDRESS),
1260                pt_to_str(PT_PHYSICAL), pt_to_str(PT_DEFROUTER));
1261            (void) fprintf(fp, "\t%s\t\t%s\n", rt_to_str(RT_DEVICE),
```



```
1262                    pt_to_str(PT_MATCH));
1263            (void) fprintf(fp, "\t%s\t\t%s, %s\n", rt_to_str(RT_RCTL),
1264                pt_to_str(PT_NAME), pt_to_str(PT_VALUE));
1265            (void) fprintf(fp, "\t%s\t\t%s, %s, %s\n", rt_to_str(RT_ATTR),
1266                pt_to_str(PT_NAME), pt_to_str(PT_TYPE),
1267                pt_to_str(PT_VALUE));
1268            (void) fprintf(fp, "\t%s\t\t%s\n", rt_to_str(RT_DATASET),
1269                pt_to_str(PT_NAME));
1270            (void) fprintf(fp, "\t%s\t%s, %s\n", rt_to_str(RT_DCPU),
1271                pt_to_str(PT_NCPUS), pt_to_str(PT_IMPORTANCE));
1272            (void) fprintf(fp, "\t%s\t%s\n", rt_to_str(RT_PCAP),
1273                pt_to_str(PT_NCPUS));
1274            (void) fprintf(fp, "\t%s\t%s, %s, %s\n", rt_to_str(RT_MCAP),
1275                pt_to_str(PT_PHYSICAL), pt_to_str(PT_SWAP),
1276                pt_to_str(PT_LOCKED));
1277            (void) fprintf(fp, "\t%s\t\t%s, %s\n", rt_to_str(RT_ADMIN),
1278                pt_to_str(PT_USER), pt_to_str(PT_AUTHS));
1279        }
1280        if (need_to_close) {
1281            int status;

1283            status = pclose(fp);
1284            if (status == -1)
1285                zerr(gettext("PAGER %s close failed (%s)."),
1286                    pager, strerror(errno));
1287            else if (WIFEXITED(status) && WEXITSTATUS(status) != 0)
1288                zerr(gettext("PAGER %s exit code: %d."),
1289                    pager, WEXITSTATUS(status));
1290        }
1276        if (need_to_close)
1277            (void) pclose(fp);
1291 }
```

unchanged_portion_omitted_

```
5354 void
5355 info_func(cmd_t *cmd)
5356 {
5357        FILE *fp = stdout;
5358        boolean_t need_to_close = B_FALSE;
5359        char *pager, *space;
5360        int type;
5361        int res1, res2;
5362        uint64_t swap_limit;
5363        uint64_t locked_limit;
5364        struct stat statbuf;

5366        assert(cmd != NULL);

5368        if (initialize(B_TRUE) != Z_OK)
5369            return;

5371        /* don't page error output */
5372        if (interactive_mode) {
5373            if ((pager = getenv("PAGER")) == NULL)
5374                pager = PAGER;
5375            space = strchr(pager, ' ');
5376            if (space)
5377                *space = '\0';
5378            if (*pager == '/' && stat(pager, &statbuf) != 0) {
5379                zerr(gettext("PAGER %s does not exist (%s)."),
5380                    pager, strerror(errno));
5381            } else {
5365            if (stat(pager, &statbuf) == 0) {
5382                if (space)
5383                    *space = ' ';
5384                if ((fp = popen(pager, "w")) == NULL) {
```

```
5368                        if ((fp = popen(pager, "w")) != NULL)
5369                                need_to_close = B_TRUE;
5370                        else
5385                                fp = stdout;
5386                                zerr(gettext("PAGER %s open failed (%s)."),
5387                                    pager, strerror(errno));
5388                        } else {
5389                                need_to_close = B_TRUE;
5373                        zerr(gettext("PAGER %s does not exist (%s)."),
5374                            pager, strerror(errno));
5390                        }
5391                }

5393                setbuf(fp, NULL);
5394        }

5396        if (!global_scope) {
5397                switch (resource_scope) {
5398                case RT_FS:
5399                        output_fs(fp, &in_progress_fstab);
5400                        break;
5401                case RT_NET:
5402                        output_net(fp, &in_progress_nwiftab);
5403                        break;
5404                case RT_DEVICE:
5405                        output_dev(fp, &in_progress_devtab);
5406                        break;
5407                case RT_RCTL:
5408                        output_rctl(fp, &in_progress_rctltab);
5409                        break;
5410                case RT_ATTR:
5411                        output_attr(fp, &in_progress_attrtab);
5412                        break;
5413                case RT_DATASET:
5414                        output_ds(fp, &in_progress_dstab);
5415                        break;
5416                case RT_DCPU:
5417                        output_pset(fp, &in_progress_psettab);
5418                        break;
5419                case RT_PCAP:
5420                        output_pcap(fp);
5421                        break;
5422                case RT_MCAP:
5423                        res1 = zonecfg_get_aliased_rctl(handle, ALIAS_MAXSWAP,
5424                            &swap_limit);
5425                        res2 = zonecfg_get_aliased_rctl(handle,
5426                            ALIAS_MAXLOCKEDMEM, &locked_limit);
5427                        output_mcap(fp, &in_progress_mcaptab, res1, swap_limit,
5428                            res2, locked_limit);
5429                        break;
5430                case RT_ADMIN:
5431                        output_auth(fp, &in_progress_admintab);
5432                        break;
5433                }
5434                goto cleanup;
5435        }

5437        type = cmd->cmd_res_type;

5439        if (gz_invalid_rt_property(type)) {
5440                zerr(gettext("%s is not a valid property for the global zone."),
5441                    rt_to_str(type));
5442                goto cleanup;
5443        }

5445        if (gz_invalid_resource(type)) {
```



```
5446                zerr(gettext("%s is not a valid resource for the global zone."),
5447                    rt_to_str(type));
5448                goto cleanup;
5449        }

5451        switch (cmd->cmd_res_type) {
5452        case RT_UNKNOWN:
5453                info_zonename(handle, fp);
5454                if (!global_zone) {
5455                        info_zonepath(handle, fp);
5456                        info_brand(handle, fp);
5457                        info_autoboot(handle, fp);
5458                        info_bootargs(handle, fp);
5459                }
5460                info_pool(handle, fp);
5461                if (!global_zone) {
5462                        info_limitpriv(handle, fp);
5463                        info_sched(handle, fp);
5464                        info_iptype(handle, fp);
5465                        info_hostid(handle, fp);
5466                        info_fs_allowed(handle, fp);
5467                }
5468                info_aliased_rctl(handle, fp, ALIAS_MAXLWPS);
5469                info_aliased_rctl(handle, fp, ALIAS_MAXPROCS);
5470                info_aliased_rctl(handle, fp, ALIAS_MAXSHMMEM);
5471                info_aliased_rctl(handle, fp, ALIAS_MAXSHMIDS);
5472                info_aliased_rctl(handle, fp, ALIAS_MAXMSGIDS);
5473                info_aliased_rctl(handle, fp, ALIAS_MAXSEMIDS);
5474                info_aliased_rctl(handle, fp, ALIAS_SHARES);
5475                if (!global_zone) {
5476                        info_fs(handle, fp, cmd);
5477                        info_net(handle, fp, cmd);
5478                        info_dev(handle, fp, cmd);
5479                }
5480                info_pset(handle, fp);
5481                info_pcap(fp);
5482                info_mcap(handle, fp);
5483                if (!global_zone) {
5484                        info_attr(handle, fp, cmd);
5485                        info_ds(handle, fp, cmd);
5486                        info_auth(handle, fp, cmd);
5487                }
5488                info_rctl(handle, fp, cmd);
5489                break;
5490        case RT_ZONENAME:
5491                info_zonename(handle, fp);
5492                break;
5493        case RT_ZONEPATH:
5494                info_zonepath(handle, fp);
5495                break;
5496        case RT_BRAND:
5497                info_brand(handle, fp);
5498                break;
5499        case RT_AUTOBOOT:
5500                info_autoboot(handle, fp);
5501                break;
5502        case RT_POOL:
5503                info_pool(handle, fp);
5504                break;
5505        case RT_LIMITPRIV:
5506                info_limitpriv(handle, fp);
5507                break;
5508        case RT_BOOTARGS:
5509                info_bootargs(handle, fp);
5510                break;
5511        case RT_SCHED:
```

```
5512                 info_sched(handle, fp);
5513                 break;
5514         case RT_IPTYPE:
5515                 info_iptype(handle, fp);
5516                 break;
5517         case RT_MAXLWPS:
5518                 info_aliased_rctl(handle, fp, ALIAS_MAXLWPS);
5519                 break;
5520         case RT_MAXPROCS:
5521                 info_aliased_rctl(handle, fp, ALIAS_MAXPROCS);
5522                 break;
5523         case RT_MAXSHMMEM:
5524                 info_aliased_rctl(handle, fp, ALIAS_MAXSHMMEM);
5525                 break;
5526         case RT_MAXSHMIDS:
5527                 info_aliased_rctl(handle, fp, ALIAS_MAXSHMIDS);
5528                 break;
5529         case RT_MAXMSGIDS:
5530                 info_aliased_rctl(handle, fp, ALIAS_MAXMSGIDS);
5531                 break;
5532         case RT_MAXSEMIDS:
5533                 info_aliased_rctl(handle, fp, ALIAS_MAXSEMIDS);
5534                 break;
5535         case RT_SHARES:
5536                 info_aliased_rctl(handle, fp, ALIAS_SHARES);
5537                 break;
5538         case RT_FS:
5539                 info_fs(handle, fp, cmd);
5540                 break;
5541         case RT_NET:
5542                 info_net(handle, fp, cmd);
5543                 break;
5544         case RT_DEVICE:
5545                 info_dev(handle, fp, cmd);
5546                 break;
5547         case RT_RCTL:
5548                 info_rctl(handle, fp, cmd);
5549                 break;
5550         case RT_ATTR:
5551                 info_attr(handle, fp, cmd);
5552                 break;
5553         case RT_DATASET:
5554                 info_ds(handle, fp, cmd);
5555                 break;
5556         case RT_DCPU:
5557                 info_pset(handle, fp);
5558                 break;
5559         case RT_PCAP:
5560                 info_pcap(fp);
5561                 break;
5562         case RT_MCAP:
5563                 info_mcap(handle, fp);
5564                 break;
5565         case RT_HOSTID:
5566                 info_hostid(handle, fp);
5567                 break;
5568         case RT_ADMIN:
5569                 info_auth(handle, fp, cmd);
5570                 break;
5571         case RT_FS_ALLOWED:
5572                 info_fs_allowed(handle, fp);
5573                 break;
5574         default:
5575                 zone_perror(rt_to_str(cmd->cmd_res_type), Z_NO_RESOURCE_TYPE,
5576                     B_TRUE);
5577         }
```



```
5579 cleanup:
5580         if (need_to_close) {
5581                 int status;

5583                 status = pclose(fp);
5584                 if (status == -1)
5585                         zerr(gettext("PAGER %s close failed (%s)."),
5586                             pager, strerror(errno));
5587                 else if (WIFEXITED(status) && WEXITSTATUS(status) != 0)
5588                         zerr(gettext("PAGER %s exit code: %d."),
5589                             pager, WEXITSTATUS(status));
5590         }
5564         if (need_to_close)
5565                 (void) pclose(fp);
5591 }
______unchanged_portion_omitted_
```

```
**********************************************************
  200633 Mon Jul  7 08:39:36 2014
new/usr/src/lib/libzonecfg/common/libzonecfg.c
3347 zonecfg(1M) is confused about selection
4956 zonecfg won't use a valid pager
**********************************************************
/*
 * 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 2014 Gary Mills
 * Copyright (c) 2003, 2010, Oracle and/or its affiliates. All rights reserved.
 */

#include <libsysevent.h>
#include <pthread.h>
#include <stdlib.h>
#include <errno.h>
#include <fnmatch.h>
#include <strings.h>
#include <unistd.h>
#include <assert.h>
#include <libgen.h>
#include <libintl.h>
#include <alloca.h>
#include <ctype.h>
#include <sys/acl.h>
#include <sys/stat.h>
#include <sys/brand.h>
#include <sys/mntio.h>
#include <sys/mnttab.h>
#include <sys/nvpair.h>
#include <sys/types.h>
#include <sys/sockio.h>
#include <sys/systeminfo.h>
#include <ftw.h>
#include <pool.h>
#include <libscf.h>
#include <libproc.h>
#include <sys/priocntl.h>
#include <libuutil.h>
#include <wait.h>
#include <bsm/adt.h>
#include <auth_attr.h>
#include <auth_list.h>
#include <secdb.h>
#include <user_attr.h>
#include <prof_attr.h>
```



```
#include <arpa/inet.h>
#include <netdb.h>

#include <libxml/xmlmemory.h>
#include <libxml/parser.h>

#include <libdevinfo.h>
#include <uuid/uuid.h>
#include <dirent.h>
#include <libbrand.h>

#include <libzonecfg.h>
#include "zonecfg_impl.h"

#define _PATH_TMPFILE   "/zonecfg.XXXXXX"
#define ZONE_CB_RETRY_COUNT             10
#define ZONE_EVENT_PING_SUBCLASS        "ping"
#define ZONE_EVENT_PING_PUBLISHER       "solaris"

/* Hard-code the DTD element/attribute/entity names just once, here. */
#define DTD_ELEM_ATTR           (const xmlChar *) "attr"
#define DTD_ELEM_COMMENT        (const xmlChar *) "comment"
#define DTD_ELEM_DEVICE         (const xmlChar *) "device"
#define DTD_ELEM_FS             (const xmlChar *) "filesystem"
#define DTD_ELEM_FSOPTION       (const xmlChar *) "fsoption"
#define DTD_ELEM_NET            (const xmlChar *) "network"
#define DTD_ELEM_RCTL           (const xmlChar *) "rctl"
#define DTD_ELEM_RCTLVALUE      (const xmlChar *) "rctl-value"
#define DTD_ELEM_ZONE           (const xmlChar *) "zone"
#define DTD_ELEM_DATASET        (const xmlChar *) "dataset"
#define DTD_ELEM_TMPPOOL        (const xmlChar *) "tmp_pool"
#define DTD_ELEM_PSET           (const xmlChar *) "pset"
#define DTD_ELEM_MCAP           (const xmlChar *) "mcap"
#define DTD_ELEM_PACKAGE        (const xmlChar *) "package"
#define DTD_ELEM_OBSOLETES      (const xmlChar *) "obsoletes"
#define DTD_ELEM_DEV_PERM       (const xmlChar *) "dev-perm"
#define DTD_ELEM_ADMIN          (const xmlChar *) "admin"

#define DTD_ATTR_ACTION         (const xmlChar *) "action"
#define DTD_ATTR_ADDRESS        (const xmlChar *) "address"
#define DTD_ATTR_ALLOWED_ADDRESS        (const xmlChar *) "allowed-address"
#define DTD_ATTR_AUTOBOOT       (const xmlChar *) "autoboot"
#define DTD_ATTR_IPTYPE         (const xmlChar *) "ip-type"
#define DTD_ATTR_DEFROUTER      (const xmlChar *) "defrouter"
#define DTD_ATTR_DIR            (const xmlChar *) "directory"
#define DTD_ATTR_LIMIT          (const xmlChar *) "limit"
#define DTD_ATTR_LIMITPRIV      (const xmlChar *) "limitpriv"
#define DTD_ATTR_BOOTARGS       (const xmlChar *) "bootargs"
#define DTD_ATTR_SCHED          (const xmlChar *) "scheduling-class"
#define DTD_ATTR_MATCH          (const xmlChar *) "match"
#define DTD_ATTR_NAME           (const xmlChar *) "name"
#define DTD_ATTR_PHYSICAL       (const xmlChar *) "physical"
#define DTD_ATTR_POOL           (const xmlChar *) "pool"
#define DTD_ATTR_PRIV           (const xmlChar *) "priv"
#define DTD_ATTR_RAW            (const xmlChar *) "raw"
#define DTD_ATTR_SPECIAL        (const xmlChar *) "special"
#define DTD_ATTR_TYPE           (const xmlChar *) "type"
#define DTD_ATTR_VALUE          (const xmlChar *) "value"
#define DTD_ATTR_ZONEPATH       (const xmlChar *) "zonepath"
#define DTD_ATTR_NCPU_MIN       (const xmlChar *) "ncpu_min"
#define DTD_ATTR_NCPU_MAX       (const xmlChar *) "ncpu_max"
#define DTD_ATTR_IMPORTANCE     (const xmlChar *) "importance"
#define DTD_ATTR_PHYSCAP        (const xmlChar *) "physcap"
#define DTD_ATTR_VERSION        (const xmlChar *) "version"
#define DTD_ATTR_ID             (const xmlChar *) "id"
```

```
 127 #define DTD_ATTR_UID            (const xmlChar *) "uid"
 128 #define DTD_ATTR_GID            (const xmlChar *) "gid"
 129 #define DTD_ATTR_MODE           (const xmlChar *) "mode"
 130 #define DTD_ATTR_ACL            (const xmlChar *) "acl"
 131 #define DTD_ATTR_BRAND          (const xmlChar *) "brand"
 132 #define DTD_ATTR_HOSTID         (const xmlChar *) "hostid"
 133 #define DTD_ATTR_USER           (const xmlChar *) "user"
 134 #define DTD_ATTR_AUTHS          (const xmlChar *) "auths"
 135 #define DTD_ATTR_FS_ALLOWED     (const xmlChar *) "fs-allowed"

 137 #define DTD_ENTITY_BOOLEAN      "boolean"
 138 #define DTD_ENTITY_DEVPATH      "devpath"
 139 #define DTD_ENTITY_DRIVER       "driver"
 140 #define DTD_ENTITY_DRVMIN       "drv_min"
 141 #define DTD_ENTITY_FALSE        "false"
 142 #define DTD_ENTITY_INT          "int"
 143 #define DTD_ENTITY_STRING       "string"
 144 #define DTD_ENTITY_TRUE         "true"
 145 #define DTD_ENTITY_UINT         "uint"

 147 #define DTD_ENTITY_BOOL_LEN     6       /* "false" */

 149 #define ATTACH_FORCED   "SUNWattached.xml"

 151 #define TMP_POOL_NAME   "SUNWtmp_%s"
 152 #define MAX_TMP_POOL_NAME       (ZONENAME_MAX + 9)
 153 #define RCAP_SERVICE    "system/rcap:default"
 154 #define POOLD_SERVICE   "system/pools/dynamic:default"

 156 /*
 157  * rctl alias definitions
 158  *
 159  * This holds the alias, the full rctl name, the default priv value, action
 160  * and lower limit.  The functions that handle rctl aliases step through
 161  * this table, matching on the alias, and using the full values for setting
 162  * the rctl entry as well the limit for validation.
 163  */
 164 static struct alias {
 165         char *shortname;
 166         char *realname;
 167         char *priv;
 168         char *action;
 169         uint64_t low_limit;
 170 } aliases[] = {
______unchanged_portion_omitted_

1760 static boolean_t
1761 match_prop(xmlNodePtr cur, const xmlChar *attr, char *user_prop)
1762 {
1763         xmlChar *gotten_prop;
1764         int prop_result;

1766         gotten_prop = xmlGetProp(cur, attr);
1767         if (gotten_prop == NULL)        /* shouldn't happen */
1768                 return (B_FALSE);
1769         prop_result = xmlStrcmp(gotten_prop, (const xmlChar *) user_prop);
1770         xmlFree(gotten_prop);
1771         return ((prop_result == 0));    /* empty strings will match */
1770         return ((prop_result == 0));
1772 }
______unchanged_portion_omitted_

2242 static int
2243 zonecfg_delete_nwif_core(zone_dochandle_t handle, struct zone_nwiftab *tabptr)
2244 {
2245         xmlNodePtr cur = handle->zone_dh_cur;
```



```
2246         boolean_t addr_match, phys_match, allowed_addr_match;

2248         for (cur = cur->xmlChildrenNode; cur != NULL; cur = cur->next) {
2249                 if (xmlStrcmp(cur->name, DTD_ELEM_NET))
2250                         continue;

2252                 addr_match = match_prop(cur, DTD_ATTR_ADDRESS,
2253                     tabptr->zone_nwif_address);
2254                 allowed_addr_match = match_prop(cur, DTD_ATTR_ALLOWED_ADDRESS,
2255                     tabptr->zone_nwif_allowed_address);
2256                 phys_match = match_prop(cur, DTD_ATTR_PHYSICAL,
2257                     tabptr->zone_nwif_physical);

2259                 if (addr_match && allowed_addr_match && phys_match) {
2258                 if ((addr_match || allowed_addr_match) && phys_match) {
2260                         xmlUnlinkNode(cur);
2261                         xmlFreeNode(cur);
2262                         return (Z_OK);
2263                 }
2264         }
2265         return (Z_NO_RESOURCE_ID);
2266 }
______unchanged_portion_omitted_
```