```
**********************************************************
  192511 Tue Jul 15 11:18:26 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 /* Copied almost verbatim from libtnfctl/prb_findexec.c */
 913 static const char *
 914 exec_cat(const char *s1, const char *s2, char *si)
 915 {
 916         char               *s;
 917         /* number of characters in s2 */
 918         int                     cnt = PATH_MAX + 1;

 920         s = si;
 921         while (*s1 && *s1 != ':') {
 922                 if (cnt > 0) {
 923                         *s++ = *s1++;
 924                         cnt--;
 925                 } else
 926                         s1++;
 927         }
 928         if (si != s && cnt > 0) {
 929                 *s++ = '/';
 930                 cnt--;
 931         }
 932         while (*s2 && cnt > 0) {
 933                 *s++ = *s2++;
 934                 cnt--;
 935         }
 936         *s = '\0';
 937         return (*s1 ? ++s1 : NULL);
```

```
 938 }

 940 /* Determine that a name exists in PATH */
 941 /* Copied with changes from libtnfctl/prb_findexec.c */
 942 static int
 943 path_find(const char *name, char *ret_path)
 944 {
 945         const char       *pathstr;
 946         char             fname[PATH_MAX + 2];
 947         const char       *cp;
 948         struct stat      stat_buf;

 950         if (*name == '\0') {
 951                 return (-1);
 952         }
 953         if ((pathstr = getenv("PATH")) == NULL) {
 954                 if (geteuid() == 0 || getuid() == 0)
 955                         pathstr = "/usr/sbin:/usr/bin";
 956                 else
 957                         pathstr = "/usr/bin:";
 958         }
 959         cp = strchr(name, '/') ? (const char *) "" : pathstr;

 961         do {
 962                 cp = exec_cat(cp, name, fname);
 963                 if (stat(fname, &stat_buf) != -1) {
 964                         /* successful find of the file */
 965                         if (ret_path != NULL)
 966                                 (void) strncpy(ret_path, fname, PATH_MAX + 2);
 967                         return (0);
 968                 }
 969         } while (cp != NULL);

 971         return (-1);
 972 }

 974 static FILE *
 975 pager_open(void) {
 976         FILE *newfp;
 977         char *pager, *space;

 979         if ((pager = getenv("PAGER")) == NULL)
 980                 pager = PAGER;

 982         space = strchr(pager, ' ');
 983         if (space)
 984                 *space = '\0';
 985         if (path_find(pager, NULL) == 0) {
 986                 if (space)
 987                         *space = ' ';
 988                 if ((newfp = popen(pager, "w")) == NULL)
 989                         zerr(gettext("PAGER open failed (%s)."),
 990                             strerror(errno));
 991                 return (newfp);
 992         } else {
 993                 zerr(gettext("PAGER %s does not exist (%s)."),
 994                     pager, strerror(errno));
 995         }
 996         return (NULL);
 997 }

 999 static void
1000 pager_close(FILE *fp) {
1001         int status;

1003         status = pclose(fp);
```



```
1004         if (status == -1)
1005                 zerr(gettext("PAGER close failed (%s)."),
1006                     strerror(errno));
1007 }

1009 /*
1010  * Called with verbose TRUE when help is explicitly requested, FALSE for
1011  * unexpected errors.
1012  */

1014 void
1015 usage(boolean_t verbose, uint_t flags)
1016 {
1017         FILE *fp = verbose ? stdout : stderr;
1018         FILE *newfp;
1019         boolean_t need_to_close = B_FALSE;
 922         char *pager, *space;
1020         int i;
 924         struct stat statbuf;

1022         /* don't page error output */
1023         if (verbose && interactive_mode) {
1024                 if ((newfp = pager_open()) != NULL) {
 928                 if ((pager = getenv("PAGER")) == NULL)
 929                         pager = PAGER;

 931                 space = strchr(pager, ' ');
 932                 if (space)
 933                         *space = '\0';
 934                 if (stat(pager, &statbuf) == 0) {
 935                         if (space)
 936                                 *space = ' ';
 937                         if ((newfp = popen(pager, "w")) != NULL) {
1025                         need_to_close = B_TRUE;
1026                         fp = newfp;
1027                 }
 941                 } else {
 942                         zerr(gettext("PAGER %s does not exist (%s)."),
 943                             pager, strerror(errno));
1028         }
 945         }

1030         if (flags & HELP_META) {
1031                 (void) fprintf(fp, gettext("More help is available for the "
1032                     "following:\n"));
1033                 (void) fprintf(fp, "\n\tcommands ('%s commands')\n",
1034                     cmd_to_str(CMD_HELP));
1035                 (void) fprintf(fp, "\tsyntax ('%s syntax')\n",
1036                     cmd_to_str(CMD_HELP));
1037                 (void) fprintf(fp, "\tusage ('%s usage')\n\n",
1038                     cmd_to_str(CMD_HELP));
1039                 (void) fprintf(fp, gettext("You may also obtain help on any "
1040                     "command by typing '%s <command-name>.'\n"),
1041                     cmd_to_str(CMD_HELP));
1042         }
1043         if (flags & HELP_RES_SCOPE) {
1044                 switch (resource_scope) {
1045                 case RT_FS:
1046                         (void) fprintf(fp, gettext("The '%s' resource scope is "
1047                             "used to configure a file-system.\n"),
1048                             rt_to_str(resource_scope));
1049                         (void) fprintf(fp, gettext("Valid commands:\n"));
1050                         (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1051                             pt_to_str(PT_DIR), gettext("<path>"));
1052                         (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1053                             pt_to_str(PT_SPECIAL), gettext("<path>"));
```

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



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

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



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

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

5424 void
5425 info_func(cmd_t *cmd)
5426 {
5427        FILE *fp = stdout;
5428        boolean_t need_to_close = B_FALSE;
5346        char *pager, *space;
5429        int type;
5430        int res1, res2;
5431        uint64_t swap_limit;
5432        uint64_t locked_limit;
5351        struct stat statbuf;

5434        assert(cmd != NULL);

5436        if (initialize(B_TRUE) != Z_OK)
5437                return;

5439        /* don't page error output */
5440        if (interactive_mode) {
```



```
5441                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)
5442                        need_to_close = B_TRUE;
5443                else
5444                        fp = stdout;
5372                } else {
5373                        zerr(gettext("PAGER %s does not exist (%s)."),
5374                            pager, strerror(errno));
5375                }

5446                setbuf(fp, NULL);
5447        }

5449        if (!global_scope) {
5450                switch (resource_scope) {
5451                case RT_FS:
5452                        output_fs(fp, &in_progress_fstab);
5453                        break;
5454                case RT_NET:
5455                        output_net(fp, &in_progress_nwiftab);
5456                        break;
5457                case RT_DEVICE:
5458                        output_dev(fp, &in_progress_devtab);
5459                        break;
5460                case RT_RCTL:
5461                        output_rctl(fp, &in_progress_rctltab);
5462                        break;
5463                case RT_ATTR:
5464                        output_attr(fp, &in_progress_attrtab);
5465                        break;
5466                case RT_DATASET:
5467                        output_ds(fp, &in_progress_dstab);
5468                        break;
5469                case RT_DCPU:
5470                        output_pset(fp, &in_progress_psettab);
5471                        break;
5472                case RT_PCAP:
5473                        output_pcap(fp);
5474                        break;
5475                case RT_MCAP:
5476                        res1 = zonecfg_get_aliased_rctl(handle, ALIAS_MAXSWAP,
5477                            &swap_limit);
5478                        res2 = zonecfg_get_aliased_rctl(handle,
5479                            ALIAS_MAXLOCKEDMEM, &locked_limit);
5480                        output_mcap(fp, &in_progress_mcaptab, res1, swap_limit,
5481                            res2, locked_limit);
5482                        break;
5483                case RT_ADMIN:
5484                        output_auth(fp, &in_progress_admintab);
5485                        break;
5486                }
5487                goto cleanup;
5488        }

5490        type = cmd->cmd_res_type;

5492        if (gz_invalid_rt_property(type)) {
5493                zerr(gettext("%s is not a valid property for the global zone."),
```

```
5494                    rt_to_str(type));
5495            goto cleanup;
5496        }

5498        if (gz_invalid_resource(type)) {
5499            zerr(gettext("%s is not a valid resource for the global zone."),
5500                rt_to_str(type));
5501            goto cleanup;
5502        }

5504        switch (cmd->cmd_res_type) {
5505        case RT_UNKNOWN:
5506            info_zonename(handle, fp);
5507            if (!global_zone) {
5508                info_zonepath(handle, fp);
5509                info_brand(handle, fp);
5510                info_autoboot(handle, fp);
5511                info_bootargs(handle, fp);
5512            }
5513            info_pool(handle, fp);
5514            if (!global_zone) {
5515                info_limitpriv(handle, fp);
5516                info_sched(handle, fp);
5517                info_iptype(handle, fp);
5518                info_hostid(handle, fp);
5519                info_fs_allowed(handle, fp);
5520            }
5521            info_aliased_rctl(handle, fp, ALIAS_MAXLWPS);
5522            info_aliased_rctl(handle, fp, ALIAS_MAXPROCS);
5523            info_aliased_rctl(handle, fp, ALIAS_MAXSHMMEM);
5524            info_aliased_rctl(handle, fp, ALIAS_MAXSHMIDS);
5525            info_aliased_rctl(handle, fp, ALIAS_MAXMSGIDS);
5526            info_aliased_rctl(handle, fp, ALIAS_MAXSEMIDS);
5527            info_aliased_rctl(handle, fp, ALIAS_SHARES);
5528            if (!global_zone) {
5529                info_fs(handle, fp, cmd);
5530                info_net(handle, fp, cmd);
5531                info_dev(handle, fp, cmd);
5532            }
5533            info_pset(handle, fp);
5534            info_pcap(fp);
5535            info_mcap(handle, fp);
5536            if (!global_zone) {
5537                info_attr(handle, fp, cmd);
5538                info_ds(handle, fp, cmd);
5539                info_auth(handle, fp, cmd);
5540            }
5541            info_rctl(handle, fp, cmd);
5542            break;
5543        case RT_ZONENAME:
5544            info_zonename(handle, fp);
5545            break;
5546        case RT_ZONEPATH:
5547            info_zonepath(handle, fp);
5548            break;
5549        case RT_BRAND:
5550            info_brand(handle, fp);
5551            break;
5552        case RT_AUTOBOOT:
5553            info_autoboot(handle, fp);
5554            break;
5555        case RT_POOL:
5556            info_pool(handle, fp);
5557            break;
5558        case RT_LIMITPRIV:
5559            info_limitpriv(handle, fp);
```



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

```
5626                 break;
5627         default:
5628                 zone_perror(rt_to_str(cmd->cmd_res_type), Z_NO_RESOURCE_TYPE,
5629                     B_TRUE);
5630         }

5632 cleanup:
5633         if (need_to_close)
5634                 (void) pager_close(fp);
5565                 (void) pclose(fp);
5635 }
```

_____***unchanged_portion_omitted_***