```
**********************************************************
    4610 Mon Feb 24 16:22:34 2014
new/usr/src/cmd/zoneadm/svc-zones
2594 implement graceful shutdown for local zones in zoneadm
**********************************************************
```

```
#!/sbin/sh
#
# 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) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright 2014 Nexenta Systems, Inc. All rights reserved.

. /lib/svc/share/smf_include.sh

#
# Return a list of running, non-global zones for which a shutdown via
# "/sbin/init 0" may work (typically only Solaris zones.)
#
shutdown_zones()
{
        zoneadm list -p | nawk -F: '{
                if ($2 != "global") {
                        print $2
                }
        }'
}

[ ! -x /usr/sbin/zoneadm ] && exit 0    # SUNWzoneu not installed

if [ -z "$SMF_FMRI" ]; then
        echo "this script can only be invoked by smf(5)"
        exit $SMF_EXIT_ERR_NOSMF
fi

# Make sure working directory is / to prevent unmounting problems.
cd /
PATH=/usr/sbin:/usr/bin; export PATH

case "$1" in
'start')
        egrep -vs '^#|^global:' /etc/zones/index || exit 0  # no local zones

        #
        # Boot the installed zones for which the "autoboot" zone property is
        # set and invoke the sysboot hook for all other installed zones.
        #
        ZONES=""
        for zone in `zoneadm list -pi | nawk -F: '{
```



```
                        if ($3 == "installed") {
                                print $2
                        }
                }''; do
                zonecfg -z $zone info autoboot | grep "true" >/dev/null 2>&1
                if [ $? -eq 0 ]; then
                        [ -z "$ZONES" ] && echo "Booting zones:\c"
                        ZONES=yes
                        echo " $zone\c"
                        #
                        # zoneadmd puts itself into its own contract so
                        # this service will lose sight of it.  We don't
                        # support restart so it is OK for zoneadmd to
                        # to be in an orphaned contract.
                        #
                        zoneadm -z $zone boot &
                else
                        zoneadm -z $zone sysboot &
                fi
        done

        #
        # Wait for all zoneadm processes to finish before allowing the
        # start method to exit.
        #
        wait
        [ -n "$ZONES" ] && echo .
        ;;

'stop')
        egrep -vs '^#|^global:' /etc/zones/index || exit 0  # no local zones
        [ "`zoneadm list`" = "global" ] && exit 0   # no zones running

        SVC_TIMEOUT=`svcprop -p stop/timeout_seconds $SMF_FMRI`

        #
        # First, try shutting down any running zones for which an "init 0" may
        # work.
        #
        MAXSHUT=`expr 3 \* $SVC_TIMEOUT \/ 4` # 3/4 of time to zone shutdown
        MAXHALT=`expr $SVC_TIMEOUT \/ 4`      # rest of time goes to halt

        zonelist=`shutdown_zones`

        if [ -n "$zonelist" ]; then
                SHUTDOWN=0
                echo "Shutting down running zones (for up to $MAXSHUT" \
                    "seconds):\c"

                for zone in $zonelist; do
                        echo " $zone\c"
                        zoneadm -z $zone shutdown &
                        zlogin -S $zone /sbin/init 0 < /dev/null >&0 2>&0 &
                        SHUTDOWN=1
                done

                [ $SHUTDOWN -eq 1 ] && echo "."

                # Allow time for zones to shutdown cleanly

                while [ $MAXSHUT -gt 0 -a "`shutdown_zones`" != "" ]; do
                        MAXSHUT=`expr $MAXSHUT - 1`
                        sleep 1 # wait a bit longer
                done
        fi
```

```
127         #
128         # Second, try halting any non-global zones still running
129         #
130         WAITPIDS=""
131         for zone in `zoneadm list`; do
132                 if [ "$zone" != "global" ]; then
133                         [ -z "$WAITPIDS" ] &&
134                             echo "Zones failed to shutdown; trying to halt " \
135                             "(for up to $MAXHALT seconds):\c"
136                         echo " $zone\c"
137                         zoneadm -z $zone halt &
138                         WAITPIDS="$WAITPIDS $!"
139                 fi
140         done
141         [ ! -z "$WAITPIDS" ] && echo .

143         # Wait for the 'zoneadm halt' commands to complete.  We will let this
144         # run forever, since the restart daemon will eventually kill us off
145         # anyway if the halts do not complete after a certain period of time.
146         wait $WAITPIDS

148         # If the halts complete but a zone is still not shutdown, it might
149         # be in a state like 'shutting_down' or 'down'.  So we give it some
150         # time to come all the way down.

152         while [ $MAXHALT -gt 0 -a "`zoneadm list`" != "global" ]; do
153                 MAXHALT=`expr $MAXHALT - 1`
154                 sleep 1 # wait a bit longer
155         done

157         # If there are any remaining file systems in zones, try to unmount them.
158         umountall -Z

160         #
161         # Report on zones which failed to shutdown.
162         #
163         for zone in `zoneadm list`; do
164                 if [ "$zone" != "global" ]; then
165                         echo "Zone '$zone' failed to halt."
166                 fi
167         done
168         [ "`zoneadm list`" != "global" ] && exit 1   # zones still running
169         ;;

171 *)
172         echo "Usage: $0 { start | stop }"
173         exit 1
174         ;;
175 esac
176 exit 0
```

```
**********************************************************
  153992 Mon Feb 24 16:22:34 2014
new/usr/src/cmd/zoneadm/zoneadm.c
2594 implement graceful shutdown for local zones in zoneadm
**********************************************************
```

```
   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  */

  27 /*
  28  * zoneadm is a command interpreter for zone administration.  It is all in
  29  * C (i.e., no lex/yacc), and all the argument passing is argc/argv based.
  30  * main() calls parse_and_run() which calls cmd_match(), then invokes the
  31  * appropriate command's handler function.  The rest of the program is the
  32  * handler functions and their helper functions.
  33  *
  34  * Some of the helper functions are used largely to simplify I18N: reducing
  35  * the need for translation notes.  This is particularly true of many of
  36  * the zerror() calls: doing e.g. zerror(gettext("%s failed"), "foo") rather
  37  * than zerror(gettext("foo failed")) with a translation note indicating
  38  * that "foo" need not be translated.
  39  */

  41 #include <stdio.h>
  42 #include <errno.h>
  43 #include <unistd.h>
  44 #include <signal.h>
  45 #include <stdarg.h>
  46 #include <ctype.h>
  47 #include <stdlib.h>
  48 #include <string.h>
  49 #include <wait.h>
  50 #include <zone.h>
  51 #include <priv.h>
  52 #include <locale.h>
  53 #include <libintl.h>
  54 #include <libzonecfg.h>
  55 #include <bsm/adt.h>
  56 #include <sys/brand.h>
  57 #include <sys/param.h>
  58 #include <sys/types.h>
  59 #include <sys/stat.h>
  60 #include <sys/statvfs.h>
  61 #include <assert.h>
```



```
  62 #include <sys/sockio.h>
  63 #include <sys/mntent.h>
  64 #include <limits.h>
  65 #include <dirent.h>
  66 #include <uuid/uuid.h>
  67 #include <fcntl.h>
  68 #include <door.h>
  69 #include <macros.h>
  70 #include <libgen.h>
  71 #include <fnmatch.h>
  72 #include <sys/modctl.h>
  73 #include <libbrand.h>
  74 #include <libscf.h>
  75 #include <procfs.h>
  76 #include <strings.h>
  77 #include <pool.h>
  78 #include <sys/pool.h>
  79 #include <sys/priocntl.h>
  80 #include <sys/fsspriocntl.h>
  81 #include <libdladm.h>
  82 #include <libdllink.h>
  83 #include <pwd.h>
  84 #include <auth_list.h>
  85 #include <auth_attr.h>
  86 #include <secdb.h>

  88 #include "zoneadm.h"

  90 #define MAXARGS 8
  91 #define SOURCE_ZONE (CMD_MAX + 1)

  93 /* Reflects kernel zone entries */
  94 typedef struct zone_entry {
  95         zoneid_t        zid;
  96         char            zname[ZONENAME_MAX];
  97         char            *zstate_str;
  98         zone_state_t    zstate_num;
  99         char            zbrand[MAXNAMELEN];
 100         char            zroot[MAXPATHLEN];
 101         char            zuuid[UUID_PRINTABLE_STRING_LENGTH];
 102         zone_iptype_t   ziptype;
 103 } zone_entry_t;
______unchanged_portion_omitted_

 130 #define SHELP_HELP      "help"
 131 #define SHELP_BOOT      "boot [-- boot_arguments]"
 132 #define SHELP_HALT      "halt"
 133 #define SHELP_READY     "ready"
 134 #define SHELP_SHUTDOWN  "shutdown [-r [-- boot_arguments]]"
 135 #define SHELP_REBOOT    "reboot [-- boot_arguments]"
 136 #define SHELP_LIST      "list [-cipv]"
 137 #define SHELP_VERIFY    "verify"
 138 #define SHELP_INSTALL   "install [brand-specific args]"
 139 #define SHELP_UNINSTALL "uninstall [-F] [brand-specific args]"
 140 #define SHELP_CLONE     "clone [-m method] [-s <ZFS snapshot>] "\
 141         "[brand-specific args] zonename"
 142 #define SHELP_MOVE      "move zonepath"
 143 #define SHELP_DETACH    "detach [-n] [brand-specific args]"
 144 #define SHELP_ATTACH    "attach [-F] [-n <path>] [brand-specific args]"
 145 #define SHELP_MARK      "mark incomplete"

 147 #define EXEC_PREFIX     "exec "
 148 #define EXEC_LEN        (strlen(EXEC_PREFIX))
 149 #define RMCOMMAND       "/usr/bin/rm -rf"

 151 static int cleanup_zonepath(char *, boolean_t);
```

```
154 static int help_func(int argc, char *argv[]);
155 static int ready_func(int argc, char *argv[]);
156 static int boot_func(int argc, char *argv[]);
157 static int shutdown_func(int argc, char *argv[]);
158 static int halt_func(int argc, char *argv[]);
159 static int reboot_func(int argc, char *argv[]);
160 static int list_func(int argc, char *argv[]);
161 static int verify_func(int argc, char *argv[]);
162 static int install_func(int argc, char *argv[]);
163 static int uninstall_func(int argc, char *argv[]);
164 static int mount_func(int argc, char *argv[]);
165 static int unmount_func(int argc, char *argv[]);
166 static int clone_func(int argc, char *argv[]);
167 static int move_func(int argc, char *argv[]);
168 static int detach_func(int argc, char *argv[]);
169 static int attach_func(int argc, char *argv[]);
170 static int mark_func(int argc, char *argv[]);
171 static int apply_func(int argc, char *argv[]);
172 static int sysboot_func(int argc, char *argv[]);
173 static int sanity_check(char *zone, int cmd_num, boolean_t running,
174     boolean_t unsafe_when_running, boolean_t force);
175 static int cmd_match(char *cmd);
176 static int verify_details(int, char *argv[]);
177 static int verify_brand(zone_dochandle_t, int, char *argv[]);
178 static int invoke_brand_handler(int, char *argv[]);

179 static struct cmd cmdtab[] = {
180         { CMD_HELP,             "help",         SHELP_HELP,     help_func },
181         { CMD_BOOT,             "boot",         SHELP_BOOT,     boot_func },
182         { CMD_HALT,             "halt",         SHELP_HALT,     halt_func },
183         { CMD_READY,            "ready",        SHELP_READY,    ready_func },
184         { CMD_SHUTDOWN,         "shutdown",     SHELP_SHUTDOWN, shutdown_func },
185         { CMD_REBOOT,           "reboot",       SHELP_REBOOT,   reboot_func },
186         { CMD_LIST,             "list",         SHELP_LIST,     list_func },
187         { CMD_VERIFY,           "verify",       SHELP_VERIFY,   verify_func },
188         { CMD_INSTALL,          "install",      SHELP_INSTALL,  install_func },
189         { CMD_UNINSTALL,        "uninstall",    SHELP_UNINSTALL,
190             uninstall_func },
191         /* mount and unmount are private commands for admin/install */
192         { CMD_MOUNT,            "mount",        NULL,           mount_func },
193         { CMD_UNMOUNT,          "unmount",      NULL,           unmount_func },
194         { CMD_CLONE,            "clone",        SHELP_CLONE,    clone_func },
195         { CMD_MOVE,             "move",         SHELP_MOVE,     move_func },
196         { CMD_DETACH,           "detach",       SHELP_DETACH,   detach_func },
197         { CMD_ATTACH,           "attach",       SHELP_ATTACH,   attach_func },
198         { CMD_MARK,             "mark",         SHELP_MARK,     mark_func },
199         { CMD_APPLY,            "apply",        NULL,           apply_func },
200         { CMD_SYSBOOT,          "sysboot",      NULL,           sysboot_func }
201 };
```

*unchanged_portion_omitted_*

```
222 /* This is a separate function because of gettext() wrapping. */
223 static char *
224 long_help(int cmd_num)
225 {
226         assert(cmd_num >= CMD_MIN && cmd_num <= CMD_MAX);
227         switch (cmd_num) {
228         case CMD_HELP:
229                 return (gettext("Print usage message."));
230         case CMD_BOOT:
231                 return (gettext("Activates (boots) specified zone.  See "
232                     "zoneadm(1m) for valid boot\n\targuments."));
233         case CMD_HALT:
234                 return (gettext("Halts specified zone, bypassing shutdown "
```



```
235                     "scripts and removing runtime\n\tresources of the zone."));
236         case CMD_READY:
237                 return (gettext("Prepares a zone for running applications but "
238                     "does not start any user\n\tprocesses in the zone."));
239         case CMD_SHUTDOWN:
240                 return (gettext("Gracefully shutdown the zone or reboot if "
241                     "the '-r' option is specified.\n\t"
242                     "See zoneadm(1m) for valid boot arguments."));
243         case CMD_REBOOT:
244                 return (gettext("Restarts the zone (equivalent to a halt / "
245                     "boot sequence).\n\tFails if the zone is not active.  "
246                     "See zoneadm(1m) for valid boot\n\targuments."));
247         case CMD_LIST:
248                 return (gettext("Lists the current zones, or a "
249                     "specific zone if indicated.  By default,\n\tall "
250                     "running zones are listed, though this can be "
251                     "expanded to all\n\tinstalled zones with the -i "
252                     "option or all configured zones with the\n\t-c "
253                     "option.  When used with the general -z <zone> and/or -u "
254                     "<uuid-match>\n\toptions, lists only the specified "
255                     "matching zone, but lists it\n\tregardless of its state, "
256                     "and the -i and -c options are disallowed.  The\n\t-v "
257                     "option can be used to display verbose information: zone "
258                     "name, id,\n\tcurrent state, root directory and options.  "
259                     "The -p option can be used\n\tto request machine-parsable "
260                     "output.  The -v and -p options are mutually\n\texclusive."
261                     "  If neither -v nor -p is used, just the zone name is "
262                     "listed."));
263         case CMD_VERIFY:
264                 return (gettext("Check to make sure the configuration "
265                     "can safely be instantiated\n\ton the machine: "
266                     "physical network interfaces exist, etc."));
267         case CMD_INSTALL:
268                 return (gettext("Install the configuration on to the system.  "
269                     "All arguments are passed to the brand installation "
270                     "function;\n\tsee brands(5) for more information."));
271         case CMD_UNINSTALL:
272                 return (gettext("Uninstall the configuration from the system.  "
273                     "The -F flag can be used\n\tto force the action.  All "
274                     "other arguments are passed to the brand\n\tuninstall "
275                     "function; see brands(5) for more information."));
276         case CMD_CLONE:
277                 return (gettext("Clone the installation of another zone.  "
278                     "The -m option can be used to\n\tspecify 'copy' which "
279                     "forces a copy of the source zone.  The -s option\n\t"
280                     "can be used to specify the name of a ZFS snapshot "
281                     "that was taken from\n\ta previous clone command.  The "
282                     "snapshot will be used as the source\n\tinstead of "
283                     "creating a new ZFS snapshot.  All other arguments are "
284                     "passed\n\tto the brand clone function; see "
285                     "brands(5) for more information."));
286         case CMD_MOVE:
287                 return (gettext("Move the zone to a new zonepath."));
288         case CMD_DETACH:
289                 return (gettext("Detach the zone from the system. The zone "
290                     "state is changed to\n\t'configured' (but the files under "
291                     "the zonepath are untouched).\n\tThe zone can subsequently "
292                     "be attached, or can be moved to another\n\tsystem and "
293                     "attached there.  The -n option can be used to specify\n\t"
294                     "'no-execute' mode.  When -n is used, the information "
295                     "needed to attach\n\tthe zone is sent to standard output "
296                     "but the zone is not actually\n\tdetached.  All other "
297                     "arguments are passed to the brand detach function;\n\tsee "
298                     "brands(5) for more information."));
299         case CMD_ATTACH:
300                 return (gettext("Attach the zone to the system.  The zone "
```

```
 301                     "state must be 'configured'\n\tprior to attach; upon "
 302                     "successful completion, the zone state will be\n\t"
 303                     "'installed'.  The system software on the current "
 304                     "system must be\n\tcompatible with the software on the "
 305                     "zone's original system.\n\tSpecify -F "
 306                     "to force the attach and skip software compatibility "
 307                     "tests.\n\tThe -n option can be used to specify "
 308                     "'no-execute' mode.  When -n is\n\tused, the information "
 309                     "needed to attach the zone is read from the\n\tspecified "
 310                     "path and the configuration is only validated.  The path "
 311                     "can\n\tbe '-' to specify standard input.  The -F and -n "
 312                     "options are mutually\n\texclusive.  All other arguments "
 313                     "are passed to the brand attach\n\tfunction; see "
 314                     "brands(5) for more information."));
 315         case CMD_MARK:
 316                 return (gettext("Set the state of the zone.  This can be used "
 317                     "to force the zone\n\tstate to 'incomplete' "
 318                     "administratively if some activity has rendered\n\tthe "
 319                     "zone permanently unusable.  The only valid state that "
 320                     "may be\n\tspecified is 'incomplete'."));
 321         default:
 322                 return ("");
 323         }
 324         /* NOTREACHED */
 325         return (NULL);
 326 }
```

*unchanged_portion_omitted_*

```
1530 static int
1531 auth_check(char *user, char *zone, int cmd_num)
1532 {
1533         char authname[MAXAUTHS];

1535         switch (cmd_num) {
1536         case CMD_LIST:
1537         case CMD_HELP:
1538                 return (Z_OK);
1539         case SOURCE_ZONE:
1540                 (void) strlcpy(authname, ZONE_CLONEFROM_AUTH, MAXAUTHS);
1541                 break;
1542         case CMD_BOOT:
1543         case CMD_HALT:
1544         case CMD_READY:
1545         case CMD_SHUTDOWN:
1546         case CMD_REBOOT:
1547         case CMD_SYSBOOT:
1548         case CMD_VERIFY:
1549         case CMD_INSTALL:
1550         case CMD_UNINSTALL:
1551         case CMD_MOUNT:
1552         case CMD_UNMOUNT:
1553         case CMD_CLONE:
1554         case CMD_MOVE:
1555         case CMD_DETACH:
1556         case CMD_ATTACH:
1557         case CMD_MARK:
1558         case CMD_APPLY:
1559         default:
1560                 (void) strlcpy(authname, ZONE_MANAGE_AUTH, MAXAUTHS);
1561                 break;
1562         }
1563         (void) strlcat(authname, KV_OBJECT, MAXAUTHS);
1564         (void) strlcat(authname, zone, MAXAUTHS);
1565         if (chkauthattr(authname, user) == 0) {
1566                 return (Z_ERR);
1567         } else {
```



```
1568                 /*
1569                  * Some subcommands, e.g. install, run subcommands,
1570                  * e.g. sysidcfg, that require a real uid of root,
1571                  *  so switch to root, here.
1572                  */
1573                 if (setuid(0) == -1) {
1574                         zperror(gettext("insufficient privilege"), B_TRUE);
1575                         return (Z_ERR);
1576                 }
1577                 return (Z_OK);
1578         }
1579 }

1581 /*
1582  * Various sanity checks; make sure:
1583  * 1. We're in the global zone.
1584  * 2. The calling user has sufficient privilege.
1585  * 3. The target zone is neither the global zone nor anything starting with
1586  *    "SUNW".
1587  * 4a. If we're looking for a 'not running' (i.e., configured or installed)
1588  *     zone, the name service knows about it.
1589  * 4b. For some operations which expect a zone not to be running, that it is
1590  *     not already running (or ready).
1591  */
1592 static int
1593 sanity_check(char *zone, int cmd_num, boolean_t running,
1594     boolean_t unsafe_when_running, boolean_t force)
1595 {
1596         zone_entry_t *zent;
1597         priv_set_t *privset;
1598         zone_state_t state, min_state;
1599         char kernzone[ZONENAME_MAX];
1600         FILE *fp;

1602         if (getzoneid() != GLOBAL_ZONEID) {
1603                 switch (cmd_num) {
1604                 case CMD_HALT:
1605                         zerror(gettext("use %s to %s this zone."), "halt(1M)",
1606                             cmd_to_str(cmd_num));
1607                         break;
1608                 case CMD_SHUTDOWN:
1609                         zerror(gettext("use %s to %s this zone."),
1610                             "shutdown(1M)", cmd_to_str(cmd_num));
1611                         break;
1612                 case CMD_REBOOT:
1613                         zerror(gettext("use %s to %s this zone."),
1614                             "reboot(1M)", cmd_to_str(cmd_num));
1615                         break;
1616                 default:
1617                         zerror(gettext("must be in the global zone to %s a "
1618                             "zone."), cmd_to_str(cmd_num));
1619                         break;
1620                 }
1621                 return (Z_ERR);
1622         }

1624         if ((privset = priv_allocset()) == NULL) {
1625                 zerror(gettext("%s failed"), "priv_allocset");
1626                 return (Z_ERR);
1627         }

1629         if (getppriv(PRIV_EFFECTIVE, privset) != 0) {
1630                 zerror(gettext("%s failed"), "getppriv");
1631                 priv_freeset(privset);
1632                 return (Z_ERR);
1633         }
```

```
1635         if (priv_isfullset(privset) == B_FALSE) {
1636                 zerror(gettext("only a privileged user may %s a zone."),
1637                     cmd_to_str(cmd_num));
1638                 priv_freeset(privset);
1639                 return (Z_ERR);
1640         }
1641         priv_freeset(privset);

1643         if (zone == NULL) {
1644                 zerror(gettext("no zone specified"));
1645                 return (Z_ERR);
1646         }

1648         if (auth_check(username, zone, cmd_num) == Z_ERR) {
1649                 zerror(gettext("User %s is not authorized to %s this zone."),
1650                     username, cmd_to_str(cmd_num));
1651                 return (Z_ERR);
1652         }

1654         if (strcmp(zone, GLOBAL_ZONENAME) == 0) {
1655                 zerror(gettext("%s operation is invalid for the global zone."),
1656                     cmd_to_str(cmd_num));
1657                 return (Z_ERR);
1658         }

1660         if (strncmp(zone, "SUNW", 4) == 0) {
1661                 zerror(gettext("%s operation is invalid for zones starting "
1662                     "with SUNW."), cmd_to_str(cmd_num));
1663                 return (Z_ERR);
1664         }

1666         if (!zonecfg_in_alt_root()) {
1667                 zent = lookup_running_zone(zone);
1668         } else if ((fp = zonecfg_open_scratch("", B_FALSE)) == NULL) {
1669                 zent = NULL;
1670         } else {
1671                 if (zonecfg_find_scratch(fp, zone, zonecfg_get_root(),
1672                     kernzone, sizeof (kernzone)) == 0)
1673                         zent = lookup_running_zone(kernzone);
1674                 else
1675                         zent = NULL;
1676                 zonecfg_close_scratch(fp);
1677         }

1679         /*
1680          * Look up from the kernel for 'running' zones.
1681          */
1682         if (running && !force) {
1683                 if (zent == NULL) {
1684                         zerror(gettext("not running"));
1685                         return (Z_ERR);
1686                 }
1687         } else {
1688                 int err;

1690                 if (unsafe_when_running && zent != NULL) {
1691                         /* check whether the zone is ready or running */
1692                         if ((err = zone_get_state(zent->zname,
1693                             &zent->zstate_num)) != Z_OK) {
1694                                 errno = err;
1695                                 zperror2(zent->zname,
1696                                     gettext("could not get state"));
1697                                 /* can't tell, so hedge */
1698                                 zent->zstate_str = "ready/running";
1699                         } else {
```



```
1700                                 zent->zstate_str =
1701                                     zone_state_str(zent->zstate_num);
1702                         }
1703                         zerror(gettext("%s operation is invalid for %s zones."),
1704                             cmd_to_str(cmd_num), zent->zstate_str);
1705                         return (Z_ERR);
1706                 }
1707                 if ((err = zone_get_state(zone, &state)) != Z_OK) {
1708                         errno = err;
1709                         zperror2(zone, gettext("could not get state"));
1710                         return (Z_ERR);
1711                 }
1712                 switch (cmd_num) {
1713                 case CMD_UNINSTALL:
1714                         if (state == ZONE_STATE_CONFIGURED) {
1715                                 zerror(gettext("is already in state '%s'."),
1716                                     zone_state_str(ZONE_STATE_CONFIGURED));
1717                                 return (Z_ERR);
1718                         }
1719                         break;
1720                 case CMD_ATTACH:
1721                         if (state == ZONE_STATE_INSTALLED) {
1722                                 zerror(gettext("is already %s."),
1723                                     zone_state_str(ZONE_STATE_INSTALLED));
1724                                 return (Z_ERR);
1725                         } else if (state == ZONE_STATE_INCOMPLETE && !force) {
1726                                 zerror(gettext("zone is %s; %s required."),
1727                                     zone_state_str(ZONE_STATE_INCOMPLETE),
1728                                     cmd_to_str(CMD_UNINSTALL));
1729                                 return (Z_ERR);
1730                         }
1731                         break;
1732                 case CMD_CLONE:
1733                 case CMD_INSTALL:
1734                         if (state == ZONE_STATE_INSTALLED) {
1735                                 zerror(gettext("is already %s."),
1736                                     zone_state_str(ZONE_STATE_INSTALLED));
1737                                 return (Z_ERR);
1738                         } else if (state == ZONE_STATE_INCOMPLETE) {
1739                                 zerror(gettext("zone is %s; %s required."),
1740                                     zone_state_str(ZONE_STATE_INCOMPLETE),
1741                                     cmd_to_str(CMD_UNINSTALL));
1742                                 return (Z_ERR);
1743                         }
1744                         break;
1745                 case CMD_DETACH:
1746                 case CMD_MOVE:
1747                 case CMD_READY:
1748                 case CMD_BOOT:
1749                 case CMD_MOUNT:
1750                 case CMD_MARK:
1751                         if ((cmd_num == CMD_BOOT || cmd_num == CMD_MOUNT) &&
1752                             force)
1753                                 min_state = ZONE_STATE_INCOMPLETE;
1754                         else if (cmd_num == CMD_MARK)
1755                                 min_state = ZONE_STATE_CONFIGURED;
1756                         else
1757                                 min_state = ZONE_STATE_INSTALLED;

1759                         if (state < min_state) {
1760                                 zerror(gettext("must be %s before %s."),
1761                                     zone_state_str(min_state),
1762                                     cmd_to_str(cmd_num));
1763                                 return (Z_ERR);
1764                         }
1765                         break;
```

```
1766                 case CMD_VERIFY:
1767                         if (state == ZONE_STATE_INCOMPLETE) {
1768                                 zerror(gettext("zone is %s; %s required."),
1769                                     zone_state_str(ZONE_STATE_INCOMPLETE),
1770                                     cmd_to_str(CMD_UNINSTALL));
1771                                 return (Z_ERR);
1772                         }
1773                         break;
1774                 case CMD_UNMOUNT:
1775                         if (state != ZONE_STATE_MOUNTED) {
1776                                 zerror(gettext("must be %s before %s."),
1777                                     zone_state_str(ZONE_STATE_MOUNTED),
1778                                     cmd_to_str(cmd_num));
1779                                 return (Z_ERR);
1780                         }
1781                         break;
1782                 case CMD_SYSBOOT:
1783                         if (state != ZONE_STATE_INSTALLED) {
1784                                 zerror(gettext("%s operation is invalid for %s "
1785                                     "zones."), cmd_to_str(cmd_num),
1786                                     zone_state_str(state));
1787                                 return (Z_ERR);
1788                         }
1789                         break;
1790                 }
1791         }
1792         return (Z_OK);
1793 }
```

*unchanged_portion_omitted_*

```
1841 static int
1842 shutdown_func(int argc, char *argv[])
1843 {
1844         zone_cmd_arg_t zarg;
1845         int arg;
1846         boolean_t reboot = B_FALSE;

1848         zarg.cmd = Z_SHUTDOWN;

1850         if (zonecfg_in_alt_root()) {
1851                 zerror(gettext("cannot shut down zone in alternate root"));
1852                 return (Z_ERR);
1853         }

1855         optind = 0;
1856         while ((arg = getopt(argc, argv, "?r")) != EOF) {
1857                 switch (arg) {
1858                 case '?':
1859                         sub_usage(SHELP_SHUTDOWN, CMD_SHUTDOWN);
1860                         return (optopt == '?' ? Z_OK : Z_USAGE);
1861                 case 'r':
1862                         reboot = B_TRUE;
1863                         break;
1864                 default:
1865                         sub_usage(SHELP_SHUTDOWN, CMD_SHUTDOWN);
1866                         return (Z_USAGE);
1867                 }
1868         }

1870         zarg.bootbuf[0] = '\0';
1871         for (; optind < argc; optind++) {
1872                 if (strlcat(zarg.bootbuf, argv[optind],
1873                     sizeof (zarg.bootbuf)) >= sizeof (zarg.bootbuf)) {
1874                         zerror(gettext("Boot argument list too long"));
1875                         return (Z_ERR);
1876                 }
```



```
1877                 if (optind < argc - 1)
1878                         if (strlcat(zarg.bootbuf, " ", sizeof (zarg.bootbuf)) >=
1879                             sizeof (zarg.bootbuf)) {
1880                                 zerror(gettext("Boot argument list too long"));
1881                                 return (Z_ERR);
1882                         }
1883         }

1885         /*
1886          * zoneadmd should be the one to decide whether or not to proceed,
1887          * so even though it seems that the third parameter below should
1888          * perhaps be B_TRUE, it really shouldn't be.
1889          */
1890         if (sanity_check(target_zone, CMD_SHUTDOWN, B_TRUE, B_FALSE, B_FALSE)
1891             != Z_OK)
1892                 return (Z_ERR);

1894         if (zonecfg_call_zoneadmd(target_zone, &zarg, locale, B_TRUE) != Z_OK)
1895                 return (Z_ERR);

1897         if (reboot) {
1898                 if (sanity_check(target_zone, CMD_BOOT, B_FALSE, B_FALSE,
1899                     B_FALSE) != Z_OK)
1900                         return (Z_ERR);

1902                 zarg.cmd = Z_BOOT;
1903                 if (zonecfg_call_zoneadmd(target_zone, &zarg, locale,
1904                     B_TRUE) != Z_OK)
1905                         return (Z_ERR);
1906         }
1907         return (Z_OK);
1908 }

1910 static int
1911 reboot_func(int argc, char *argv[])
1912 {
1913         zone_cmd_arg_t zarg;
1914         int arg;

1916         if (zonecfg_in_alt_root()) {
1917                 zerror(gettext("cannot reboot zone in alternate root"));
1918                 return (Z_ERR);
1919         }

1921         optind = 0;
1922         if ((arg = getopt(argc, argv, "?")) != EOF) {
1923                 switch (arg) {
1924                 case '?':
1925                         sub_usage(SHELP_REBOOT, CMD_REBOOT);
1926                         return (optopt == '?' ? Z_OK : Z_USAGE);
1927                 default:
1928                         sub_usage(SHELP_REBOOT, CMD_REBOOT);
1929                         return (Z_USAGE);
1930                 }
1931         }

1933         zarg.bootbuf[0] = '\0';
1934         for (; optind < argc; optind++) {
1935                 if (strlcat(zarg.bootbuf, argv[optind],
1936                     sizeof (zarg.bootbuf)) >= sizeof (zarg.bootbuf)) {
1937                         zerror(gettext("Boot argument list too long"));
1938                         return (Z_ERR);
1939                 }
1940                 if (optind < argc - 1)
1941                         if (strlcat(zarg.bootbuf, " ", sizeof (zarg.bootbuf)) >=
1942                             sizeof (zarg.bootbuf)) {
```

```
1943                                zerror(gettext("Boot argument list too long"));
1944                                return (Z_ERR);
1945                        }
1946        }


1949        /*
1950         * zoneadmd should be the one to decide whether or not to proceed,
1951         * so even though it seems that the fourth parameter below should
1952         * perhaps be B_TRUE, it really shouldn't be.
1953         */
1954        if (sanity_check(target_zone, CMD_REBOOT, B_TRUE, B_FALSE, B_FALSE)
1955            != Z_OK)
1956                return (Z_ERR);
1957        if (verify_details(CMD_REBOOT, argv) != Z_OK)
1958                return (Z_ERR);

1960        zarg.cmd = Z_REBOOT;
1961        return ((zonecfg_call_zoneadmd(target_zone, &zarg, locale, B_TRUE) == 0)
1962            ? Z_OK : Z_ERR);
1963 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    4061 Mon Feb 24 16:22:34 2014
new/usr/src/cmd/zoneadm/zoneadm.h
2594 implement graceful shutdown for local zones in zoneadm
**********************************************************
```

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

#ifndef _ZONEADM_H
#define _ZONEADM_H

#include <sys/types.h>

#define CMD_HELP        0
#define CMD_BOOT        1
#define CMD_HALT        2
#define CMD_READY       3
#define CMD_SHUTDOWN    4
#define CMD_REBOOT      5
#define CMD_LIST        6
#define CMD_VERIFY      7
#define CMD_INSTALL     8
#define CMD_UNINSTALL   9
#define CMD_MOUNT       10
#define CMD_UNMOUNT     11
#define CMD_CLONE       12
#define CMD_MOVE        13
#define CMD_DETACH      14
#define CMD_ATTACH      15
#define CMD_MARK        16
#define CMD_APPLY       17
#define CMD_SYSBOOT     18
#define CMD_REBOOT      4
#define CMD_LIST        5
#define CMD_VERIFY      6
#define CMD_INSTALL     7
#define CMD_UNINSTALL   8
#define CMD_MOUNT       9
#define CMD_UNMOUNT     10
#define CMD_CLONE       11
#define CMD_MOVE        12
#define CMD_DETACH      13
```



```
#define CMD_ATTACH      14
#define CMD_MARK        15
#define CMD_APPLY       16
#define CMD_SYSBOOT     17

#define CMD_MIN         CMD_HELP
#define CMD_MAX         CMD_SYSBOOT

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

#define Z_ERR           1
#define Z_USAGE         2
#define Z_FATAL         3

#define SW_CMP_NONE     0x0
#define SW_CMP_SRC      0x01
#define SW_CMP_SILENT   0x02

/*
 * This structure stores information about mounts of interest within an
 * installed zone.
 */
typedef struct zone_mounts {
        /* The zone's zonepath */
        char            *zonepath;

        /* The length of zonepath */
        int             zonepath_len;

        /*
         * This indicates the number of unexpected mounts that were encountered
         * in the zone.
         */
        int             num_unexpected_mounts;

        /*
         * This is the number of overlay mounts detected on the zone's root
         * directory.
         */
        int             num_root_overlay_mounts;

        /*
         * This is used to track important zone root mount information.  The
         * mnt_time field isn't used.  If root_mnttab is NULL, then the
         * associated zone doesn't have a mounted root filesystem.
         *
         * NOTE: mnt_mountp is non-NULL iff the zone's root filesystem is a
         * ZFS filesystem with a non-legacy mountpoint.  In this case, it
         * refers to a string containing the dataset's mountpoint.
         */
        struct mnttab   *root_mnttab;
} zone_mounts_t;
______unchanged_portion_omitted_
```

```
**********************************************************
    2811 Mon Feb 24 16:22:34 2014
new/usr/src/cmd/zoneadm/zones.xml
2594 implement graceful shutdown for local zones in zoneadm
**********************************************************
```

```
<?xml version="1.0"?>
<!DOCTYPE service_bundle SYSTEM "/usr/share/lib/xml/dtd/service_bundle.dtd.1">
<!--
 Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 Use is subject to license terms.

 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

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

    NOTE:  This service manifest is not editable; its contents will
    be overwritten by package or patch operations, including
    operating system upgrade.  Make customizations in a different
    file.

 Copyright 2014 Nexenta Systems, Inc. All rights reserved.
-->

<service_bundle type='manifest' name='SUNWzoner:zones'>

<!--
        The only effect of this service is to causes zones with the
        "autoboot" property set to "true" to boot at system startup.
-->
<service
        name='system/zones'
        type='service'
        version='1'>

        <create_default_instance enabled='false' />

        <single_instance />

        <dependency
                name='multi-user-server'
                type='service'
                grouping='require_all'
                restart_on='none'>
                <service_fmri value='svc:/milestone/multi-user-server' />
        </dependency>

        <exec_method
                type='method'
                name='start'
```



```
                exec='/lib/svc/method/svc-zones %m'
                timeout_seconds='60'>
        </exec_method>

        <!--
            The stop method reads the timeout_seconds property and
            spends 3/4 of the allotted time waiting for zones to
            cleanly shut down. If some zones don't shutdown after
            the 3/4 time has elapsed, the method spends the remaining
            1/4 trying to more forcibly halt the zones
            cleanly shut down (by running 'init 0' in each one).  If
            some zones don't shutdown after the 3/4 time has elapsed,
            the method spends the remaining 1/4 trying to more forcibly
            halt the zones
        -->
        <exec_method
                type='method'
                name='stop'
                exec='/lib/svc/method/svc-zones %m'
                timeout_seconds='100'>
        </exec_method>

        <property_group name='startd' type='framework'>
                <propval name='duration' type='astring' value='transient' />
        </property_group>

        <stability value='Unstable' />

        <template>
                <common_name>
                        <loctext xml:lang='C'>
                        Zones autoboot and graceful shutdown
                        </loctext>
                </common_name>
                <documentation>
                        <manpage title='zones' section='5' manpath='/usr/share/m
                        <manpage
                                title='zonecfg'
                                section='1M'
                                manpath='/usr/share/man' />
                </documentation>
        </template>
</service>

</service_bundle>
```

```
**********************************************************
    1721 Mon Feb 24 16:22:35 2014
new/usr/src/cmd/zoneadmd/Makefile
2594 implement graceful shutdown for local zones in zoneadm
**********************************************************
```

```
#
# 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) 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright 2014 Nexenta Systems, Inc. All rights reserved.
#

PROG= zoneadmd

include ../Makefile.cmd

ROOTCMDDIR=     $(ROOTLIB)/zones

OBJS= zoneadmd.o zcons.o vplat.o
SRCS = $(OBJS:.o=.c)
POFILE=zoneadmd_all.po
POFILES= $(OBJS:%.o=%.po)

CFLAGS += $(CCVERBOSE)
CERRWARN += -_gcc=-Wno-switch
CERRWARN += -_gcc=-Wno-parentheses
CERRWARN += -_gcc=-Wno-uninitialized

LDLIBS += -lsocket -lzonecfg -lnsl -ldevinfo -ldevice -lnvpair \
        -lgen -lbsm -lcontract -lzfs -luuid -lbrand -ldladm -ltsnet -ltsol \
        -linetutil -lscf
        -linetutil
XGETFLAGS += -a -x zoneadmd.xcl

.KEEP_STATE:

.PARALLEL:

all: $(PROG)

$(PROG): $(OBJS)
        $(LINK.c) -o $@ $(OBJS) $(LDLIBS)
        $(POST_PROCESS)

install: all $(ROOTCMD)
```



```
$(POFILE): $(POFILES)
        $(RM) $@
        $(CAT) $(POFILES) > $@

clean:
        $(RM) $(OBJS)

lint:   lint_SRCS

check:
        $(CSTYLE) -p -P $(SRCS:%=%)

include ../Makefile.targ
```

```
**********************************************************
   59797 Mon Feb 24 16:22:35 2014
new/usr/src/cmd/zoneadmd/zoneadmd.c
2594 implement graceful shutdown for local zones in zoneadm
**********************************************************
/*
 * 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.
 */

/*
 * zoneadmd manages zones; one zoneadmd process is launched for each
 * non-global zone on the system.  This daemon juggles four jobs:
 *
 * - Implement setup and teardown of the zone "virtual platform": mount and
 *   unmount filesystems; create and destroy network interfaces; communicate
 *   with devfsadmd to lay out devices for the zone; instantiate the zone
 *   console device; configure process runtime attributes such as resource
 *   controls, pool bindings, fine-grained privileges.
 *
 * - Launch the zone's init(1M) process.
 *
 * - Implement a door server; clients (like zoneadm) connect to the door
 *   server and request zone state changes.  The kernel is also a client of
 *   this door server.  A request to halt or reboot the zone which originates
 *   *inside* the zone results in a door upcall from the kernel into zoneadmd.
 *
 *   One minor problem is that messages emitted by zoneadmd need to be passed
 *   back to the zoneadm process making the request.  These messages need to
 *   be rendered in the client's locale; so, this is passed in as part of the
 *   request.  The exception is the kernel upcall to zoneadmd, in which case
 *   messages are syslog'd.
 *
 *   To make all of this work, the Makefile adds -a to xgettext to extract *all*
 *   strings, and an exclusion file (zoneadmd.xcl) is used to exclude those
 *   strings which do not need to be translated.
 *
 * - Act as a console server for zlogin -C processes; see comments in zcons.c
 *   for more information about the zone console architecture.
 *
 * DESIGN NOTES
 *
 * Restart:
 *   A chief design constraint of zoneadmd is that it should be restartable in
 *   the case that the administrator kills it off, or it suffers a fatal error,
```



```
 *   without the running zone being impacted; this is akin to being able to
 *   reboot the service processor of a server without affecting the OS instance.
 */

#include <sys/param.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/sysmacros.h>

#include <bsm/adt.h>
#include <bsm/adt_event.h>

#include <alloca.h>
#include <assert.h>
#include <errno.h>
#include <door.h>
#include <fcntl.h>
#include <locale.h>
#include <signal.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <synch.h>
#include <syslog.h>
#include <thread.h>
#include <unistd.h>
#include <wait.h>
#include <limits.h>
#include <zone.h>
#include <libbrand.h>
#include <sys/brand.h>
#include <libcontract.h>
#include <libcontract_priv.h>
#include <sys/brand.h>
#include <sys/contract/process.h>
#include <sys/ctfs.h>
#include <libdladm.h>
#include <sys/dls_mgmt.h>
#include <libscf.h>

#include <libzonecfg.h>
#include <zonestat_impl.h>
#include "zoneadmd.h"

static char *progname;
char *zone_name;        /* zone which we are managing */
char pool_name[MAXNAMELEN];
char default_brand[MAXNAMELEN];
char brand_name[MAXNAMELEN];
boolean_t zone_isnative;
boolean_t zone_iscluster;
boolean_t zone_islabeled;
boolean_t shutdown_in_progress;
static zoneid_t zone_id;
dladm_handle_t dld_handle = NULL;

static char pre_statechg_hook[2 * MAXPATHLEN];
static char post_statechg_hook[2 * MAXPATHLEN];
char query_hook[2 * MAXPATHLEN];

zlog_t logsys;

mutex_t lock = DEFAULTMUTEX;    /* to serialize stuff */
```

```
 128 mutex_t msglock = DEFAULTMUTEX; /* for calling setlocale() */

 130 static sema_t scratch_sem;      /* for scratch zones */

 132 static char     zone_door_path[MAXPATHLEN];
 133 static int      zone_door = -1;

 135 boolean_t in_death_throes = B_FALSE;    /* daemon is dying */
 136 boolean_t bringup_failure_recovery = B_FALSE; /* ignore certain failures */

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

 142 #define DEFAULT_LOCALE  "C"

 144 static const char *
 145 z_cmd_name(zone_cmd_t zcmd)
 146 {
 147         /* This list needs to match the enum in sys/zone.h */
 148         static const char *zcmdstr[] = {
 149                 "ready", "boot", "forceboot", "reboot", "halt",
 150                 "note_uninstalling", "mount", "forcemount", "unmount",
 151                 "shutdown"
 147                 "note_uninstalling", "mount", "forcemount", "unmount"
 152         };

 154         if (zcmd >= sizeof (zcmdstr) / sizeof (*zcmdstr))
 155                 return ("unknown");
 156         else
 157                 return (zcmdstr[(int)zcmd]);
 158 }
```

*unchanged_portion_omitted_*

```
 993 static int
 994 zone_graceful_shutdown(zlog_t *zlogp)
 995 {
 996         zoneid_t zoneid;
 997         pid_t child;
 998         char cmdbuf[MAXPATHLEN];
 999         brand_handle_t bh = NULL;
1000         char zpath[MAXPATHLEN];
1001         ctid_t ct;
1002         int tmpl_fd;
1003         int child_status;

1005         if (shutdown_in_progress) {
1006                 zerror(zlogp, B_FALSE, "shutdown already in progress");
1007                 return (-1);
1008         }

1010         if ((zoneid = getzoneidbyname(zone_name)) == -1) {
1011                 zerror(zlogp, B_TRUE, "unable to get zoneid");
1012                 return (-1);
1013         }

1015         /* Get a handle to the brand info for this zone */
1016         if ((bh = brand_open(brand_name)) == NULL) {
1017                 zerror(zlogp, B_FALSE, "unable to determine zone brand");
1018                 return (-1);
1019         }

1021         if (zone_get_zonepath(zone_name, zpath, sizeof (zpath)) != Z_OK) {
1022                 zerror(zlogp, B_FALSE, "unable to determine zone path");
1023                 brand_close(bh);
1024                 return (-1);
```



```
1025         }

1027         /*
1028          * If there is a brand 'shutdown' callback, execute it now to give the
1029          * brand a chance to cleanup any custom configuration.
1030          */
1031         (void) strcpy(cmdbuf, EXEC_PREFIX);
1032         if (brand_get_shutdown(bh, zone_name, zpath, cmdbuf + EXEC_LEN,
1033             sizeof (cmdbuf) - EXEC_LEN) != 0 || strlen(cmdbuf) <= EXEC_LEN) {
1034                 (void) strcat(cmdbuf, SHUTDOWN_DEFAULT);
1035         }
1036         brand_close(bh);

1038         if ((tmpl_fd = init_template()) == -1) {
1039                 zerror(zlogp, B_TRUE, "failed to create contract");
1040                 return (-1);
1041         }

1043         if ((child = fork()) == -1) {
1044                 (void) ct_tmpl_clear(tmpl_fd);
1045                 (void) close(tmpl_fd);
1046                 zerror(zlogp, B_TRUE, "failed to fork");
1047                 return (-1);
1048         } else if (child == 0) {
1049                 (void) ct_tmpl_clear(tmpl_fd);
1050                 if (zone_enter(zoneid) == -1) {
1051                         _exit(errno);
1052                 }
1053                 _exit(execl("/bin/sh", "sh", "-c", cmdbuf, (char *)NULL));
1054         }

1056         if (contract_latest(&ct) == -1)
1057                 ct = -1;
1058         (void) ct_tmpl_clear(tmpl_fd);
1059         (void) close(tmpl_fd);

1061         if (waitpid(child, &child_status, 0) != child) {
1062                 /* unexpected: we must have been signalled */
1063                 (void) contract_abandon_id(ct);
1064                 return (-1);
1065         }

1067         (void) contract_abandon_id(ct);
1068         if (WEXITSTATUS(child_status) != 0) {
1069                 errno = WEXITSTATUS(child_status);
1070                 zerror(zlogp, B_FALSE, "unable to shutdown zone");
1071                 return (-1);
1072         }

1074         shutdown_in_progress = B_TRUE;

1076         return (0);
1077 }

1079 static int
1080 zone_wait_shutdown(zlog_t *zlogp)
1081 {
1082         zone_state_t zstate;
1083         uint64_t *tm = NULL;
1084         scf_simple_prop_t *prop = NULL;
1085         int timeout;
1086         int tries;
1087         int rc = -1;

1089         /* Get default stop timeout from SMF framework */
1090         timeout = SHUTDOWN_WAIT;
```

```
1091         if ((prop = scf_simple_prop_get(NULL, SHUTDOWN_FMRI, "stop",
1092             SCF_PROPERTY_TIMEOUT)) != NULL) {
1093                 if ((tm = scf_simple_prop_next_count(prop)) != NULL) {
1094                         if (tm != 0)
1095                                 timeout = *tm;
1096                 }
1097                 scf_simple_prop_free(prop);
1098         }

1100         /* allow time for zone to shutdown cleanly */
1101         for (tries = 0; tries < timeout; tries ++) {
1102                 (void) sleep(1);
1103                 if (zone_get_state(zone_name, &zstate) == Z_OK &&
1104                     zstate == ZONE_STATE_INSTALLED) {
1105                         rc = 0;
1106                         break;
1107                 }
1108         }

1110         if (rc != 0)
1111                 zerror(zlogp, B_FALSE, "unable to shutdown zone");

1113         shutdown_in_progress = B_FALSE;

1115         return (rc);
1116 }



1120 /*
1121  * Generate AUE_zone_state for a command that boots a zone.
1122  */
1123 static void
1124 audit_put_record(zlog_t *zlogp, ucred_t *uc, int return_val,
1125     char *new_state)
1126 {
1127         adt_session_data_t      *ah;
1128         adt_event_data_t        *event;
1129         int                     pass_fail, fail_reason;

1131         if (!adt_audit_enabled())
1132                 return;

1134         if (return_val == 0) {
1135                 pass_fail = ADT_SUCCESS;
1136                 fail_reason = ADT_SUCCESS;
1137         } else {
1138                 pass_fail = ADT_FAILURE;
1139                 fail_reason = ADT_FAIL_VALUE_PROGRAM;
1140         }

1142         if (adt_start_session(&ah, NULL, 0)) {
1143                 zerror(zlogp, B_TRUE, gettext("audit failure."));
1144                 return;
1145         }
1146         if (adt_set_from_ucred(ah, uc, ADT_NEW)) {
1147                 zerror(zlogp, B_TRUE, gettext("audit failure."));
1148                 (void) adt_end_session(ah);
1149                 return;
1150         }

1152         event = adt_alloc_event(ah, ADT_zone_state);
1153         if (event == NULL) {
1154                 zerror(zlogp, B_TRUE, gettext("audit failure."));
1155                 (void) adt_end_session(ah);
1156                 return;
```



```
1157         }
1158         event->adt_zone_state.zonename = zone_name;
1159         event->adt_zone_state.new_state = new_state;

1161         if (adt_put_event(event, pass_fail, fail_reason))
1162                 zerror(zlogp, B_TRUE, gettext("audit failure."));

1164         adt_free_event(event);

1166         (void) adt_end_session(ah);
1167 }

1169 /*
1170  * The main routine for the door server that deals with zone state transitions.
1171  */
1172 /* ARGSUSED */
1173 static void
1174 server(void *cookie, char *args, size_t alen, door_desc_t *dp,
1175     uint_t n_desc)
1176 {
1177         ucred_t *uc = NULL;
1178         const priv_set_t *eset;

1180         zone_state_t zstate;
1181         zone_cmd_t cmd;
1182         zone_cmd_arg_t *zargp;

1184         boolean_t kernelcall;

1186         int rval = -1;
1187         uint64_t uniqid;
1188         zoneid_t zoneid = -1;
1189         zlog_t zlog;
1190         zlog_t *zlogp;
1191         zone_cmd_rval_t *rvalp;
1192         size_t rlen = getpagesize(); /* conservative */
1193         fs_callback_t cb;
1194         brand_handle_t bh;
1195         boolean_t wait_shut = B_FALSE;

1197         /* LINTED E_BAD_PTR_CAST_ALIGN */
1198         zargp = (zone_cmd_arg_t *)args;

1200         /*
1201          * When we get the door unref message, we've fdetach'd the door, and
1202          * it is time for us to shut down zoneadmd.
1203          */
1204         if (zargp == DOOR_UNREF_DATA) {
1205                 /*
1206                  * See comment at end of main() for info on the last rites.
1207                  */
1208                 exit(0);
1209         }

1211         if (zargp == NULL) {
1212                 (void) door_return(NULL, 0, 0, 0);
1213         }

1215         rvalp = alloca(rlen);
1216         bzero(rvalp, rlen);
1217         zlog.logfile = NULL;
1218         zlog.buflen = zlog.loglen = rlen - sizeof (zone_cmd_rval_t) + 1;
1219         zlog.buf = rvalp->errbuf;
1220         zlog.log = zlog.buf;
1221         /* defer initialization of zlog.locale until after credential check */
1222         zlogp = &zlog;
```

```
1224         if (alen != sizeof (zone_cmd_arg_t)) {
1225                 /*
1226                  * This really shouldn't be happening.
1227                  */
1228                 zerror(&logsys, B_FALSE, "argument size (%d bytes) "
1229                     "unexpected (expected %d bytes)", alen,
1230                     sizeof (zone_cmd_arg_t));
1231                 goto out;
1232         }
1233         cmd = zargp->cmd;

1235         if (door_ucred(&uc) != 0) {
1236                 zerror(&logsys, B_TRUE, "door_ucred");
1237                 goto out;
1238         }
1239         eset = ucred_getprivset(uc, PRIV_EFFECTIVE);
1240         if (ucred_getzoneid(uc) != GLOBAL_ZONEID ||
1241             (eset != NULL ? !priv_ismember(eset, PRIV_SYS_CONFIG) :
1242             ucred_geteuid(uc) != 0)) {
1243                 zerror(&logsys, B_FALSE, "insufficient privileges");
1244                 goto out;
1245         }

1247         kernelcall = ucred_getpid(uc) == 0;

1249         /*
1250          * This is safe because we only use a zlog_t throughout the
1251          * duration of a door call; i.e., by the time the pointer
1252          * might become invalid, the door call would be over.
1253          */
1254         zlog.locale = kernelcall ? DEFAULT_LOCALE : zargp->locale;

1256         (void) mutex_lock(&lock);

1258         /*
1259          * Once we start to really die off, we don't want more connections.
1260          */
1261         if (in_death_throes) {
1262                 (void) mutex_unlock(&lock);
1263                 ucred_free(uc);
1264                 (void) door_return(NULL, 0, 0, 0);
1265                 thr_exit(NULL);
1266         }

1268         /*
1269          * Check for validity of command.
1270          */
1271         if (cmd != Z_READY && cmd != Z_BOOT && cmd != Z_FORCEBOOT &&
1272             cmd != Z_REBOOT && cmd != Z_SHUTDOWN && cmd != Z_HALT &&
1273             cmd != Z_NOTE_UNINSTALLING && cmd != Z_MOUNT &&
1274             cmd != Z_FORCEMOUNT && cmd != Z_UNMOUNT) {
1140             cmd != Z_REBOOT && cmd != Z_HALT && cmd != Z_NOTE_UNINSTALLING &&
1141             cmd != Z_MOUNT && cmd != Z_FORCEMOUNT && cmd != Z_UNMOUNT) {
1275                 zerror(&logsys, B_FALSE, "invalid command %d", (int)cmd);
1276                 goto out;
1277         }

1279         if (kernelcall && (cmd != Z_HALT && cmd != Z_REBOOT)) {
1280                 /*
1281                  * Can't happen
1282                  */
1283                 zerror(&logsys, B_FALSE, "received unexpected kernel upcall %d",
1284                     cmd);
1285                 goto out;
1286         }
```



```
1287         /*
1288          * We ignore the possibility of someone calling zone_create(2)
1289          * explicitly; all requests must come through zoneadmd.
1290          */
1291         if (zone_get_state(zone_name, &zstate) != Z_OK) {
1292                 /*
1293                  * Something terribly wrong happened
1294                  */
1295                 zerror(&logsys, B_FALSE, "unable to determine state of zone");
1296                 goto out;
1297         }

1299         if (kernelcall) {
1300                 /*
1301                  * Kernel-initiated requests may lose their validity if the
1302                  * zone_t the kernel was referring to has gone away.
1303                  */
1304                 if ((zoneid = getzoneidbyname(zone_name)) == -1 ||
1305                     zone_getattr(zoneid, ZONE_ATTR_UNIQID, &uniqid,
1306                     sizeof (uniqid)) == -1 || uniqid != zargp->uniqid) {
1307                         /*
1308                          * We're not talking about the same zone. The request
1309                          * must have arrived too late.  Return error.
1310                          */
1311                         rval = -1;
1312                         goto out;
1313                 }
1314                 zlogp = &logsys;        /* Log errors to syslog */
1315         }

1317         /*
1318          * If we are being asked to forcibly mount or boot a zone, we
1319          * pretend that an INCOMPLETE zone is actually INSTALLED.
1320          */
1321         if (zstate == ZONE_STATE_INCOMPLETE &&
1322             (cmd == Z_FORCEBOOT || cmd == Z_FORCEMOUNT))
1323                 zstate = ZONE_STATE_INSTALLED;

1325         switch (zstate) {
1326         case ZONE_STATE_CONFIGURED:
1327         case ZONE_STATE_INCOMPLETE:
1328                 /*
1329                  * Not our area of expertise; we just print a nice message
1330                  * and die off.
1331                  */
1332                 zerror(zlogp, B_FALSE,
1333                     "%s operation is invalid for zones in state '%s'",
1334                     z_cmd_name(cmd), zone_state_str(zstate));
1335                 break;

1337         case ZONE_STATE_INSTALLED:
1338                 switch (cmd) {
1339                 case Z_READY:
1340                         rval = zone_ready(zlogp, Z_MNT_BOOT, zstate);
1341                         if (rval == 0)
1342                                 eventstream_write(Z_EVT_ZONE_READIED);
1343                         break;
1344                 case Z_BOOT:
1345                 case Z_FORCEBOOT:
1346                         eventstream_write(Z_EVT_ZONE_BOOTING);
1347                         if ((rval = zone_ready(zlogp, Z_MNT_BOOT, zstate))
1348                             == 0) {
1349                                 rval = zone_bootup(zlogp, zargp->bootbuf,
1350                                     zstate);
1351                         }
1352                         audit_put_record(zlogp, uc, rval, "boot");
```

```
1353                        if (rval != 0) {
1354                                bringup_failure_recovery = B_TRUE;
1355                                (void) zone_halt(zlogp, B_FALSE, B_FALSE,
1356                                    zstate);
1357                                eventstream_write(Z_EVT_ZONE_BOOTFAILED);
1358                        }
1359                        break;
1360                case Z_SHUTDOWN:
1361                case Z_HALT:
1362                        if (kernelcall) /* Invalid; can't happen */
1363                                abort();
1364                        /*
1365                         * We could have two clients racing to halt this
1366                         * zone; the second client loses, but his request
1367                         * doesn't fail, since the zone is now in the desired
1368                         * state.
1369                         */
1370                        zerror(zlogp, B_FALSE, "zone is already halted");
1371                        rval = 0;
1372                        break;
1373                case Z_REBOOT:
1374                        if (kernelcall) /* Invalid; can't happen */
1375                                abort();
1376                        zerror(zlogp, B_FALSE, "%s operation is invalid "
1377                            "for zones in state '%s'", z_cmd_name(cmd),
1378                            zone_state_str(zstate));
1379                        rval = -1;
1380                        break;
1381                case Z_NOTE_UNINSTALLING:
1382                        if (kernelcall) /* Invalid; can't happen */
1383                                abort();
1384                        /*
1385                         * Tell the console to print out a message about this.
1386                         * Once it does, we will be in_death_throes.
1387                         */
1388                        eventstream_write(Z_EVT_ZONE_UNINSTALLING);
1389                        break;
1390                case Z_MOUNT:
1391                case Z_FORCEMOUNT:
1392                        if (kernelcall) /* Invalid; can't happen */
1393                                abort();
1394                        if (!zone_isnative && !zone_iscluster &&
1395                            !zone_islabeled) {
1396                                /*
1397                                 * -U mounts the zone without lofs mounting
1398                                 * zone file systems back into the scratch
1399                                 * zone.  This is required when mounting
1400                                 * non-native branded zones.
1401                                 */
1402                                (void) strlcpy(zargp->bootbuf, "-U",
1403                                    BOOTARGS_MAX);
1404                        }

1406                        rval = zone_ready(zlogp,
1407                            strcmp(zargp->bootbuf, "-U") == 0 ?
1408                            Z_MNT_UPDATE : Z_MNT_SCRATCH, zstate);
1409                        if (rval != 0)
1410                                break;

1412                        eventstream_write(Z_EVT_ZONE_READIED);

1414                        /*
1415                         * Get a handle to the default brand info.
1416                         * We must always use the default brand file system
1417                         * list when mounting the zone.
1418                         */
```



```
1419                        if ((bh = brand_open(default_brand)) == NULL) {
1420                                rval = -1;
1421                                break;
1422                        }

1424                        /*
1425                         * Get the list of filesystems to mount from
1426                         * the brand configuration.  These mounts are done
1427                         * via a thread that will enter the zone, so they
1428                         * are done from within the context of the zone.
1429                         */
1430                        cb.zlogp = zlogp;
1431                        cb.zoneid = zone_id;
1432                        cb.mount_cmd = B_TRUE;
1433                        rval = brand_platform_iter_mounts(bh,
1434                            mount_early_fs, &cb);

1436                        brand_close(bh);

1438                        /*
1439                         * Ordinarily, /dev/fd would be mounted inside the zone
1440                         * by svc:/system/filesystem/usr:default, but since
1441                         * we're not booting the zone, we need to do this
1442                         * manually.
1443                         */
1444                        if (rval == 0)
1445                                rval = mount_early_fs(&cb,
1446                                    "fd", "/dev/fd", "fd", NULL);
1447                        break;
1448                case Z_UNMOUNT:
1449                        if (kernelcall) /* Invalid; can't happen */
1450                                abort();
1451                        zerror(zlogp, B_FALSE, "zone is already unmounted");
1452                        rval = 0;
1453                        break;
1454                }
1455                break;

1457        case ZONE_STATE_READY:
1458                switch (cmd) {
1459                case Z_READY:
1460                        /*
1461                         * We could have two clients racing to ready this
1462                         * zone; the second client loses, but his request
1463                         * doesn't fail, since the zone is now in the desired
1464                         * state.
1465                         */
1466                        zerror(zlogp, B_FALSE, "zone is already ready");
1467                        rval = 0;
1468                        break;
1469                case Z_BOOT:
1470                        (void) strlcpy(boot_args, zargp->bootbuf,
1471                            sizeof (boot_args));
1472                        eventstream_write(Z_EVT_ZONE_BOOTING);
1473                        rval = zone_bootup(zlogp, zargp->bootbuf, zstate);
1474                        audit_put_record(zlogp, uc, rval, "boot");
1475                        if (rval != 0) {
1476                                bringup_failure_recovery = B_TRUE;
1477                                (void) zone_halt(zlogp, B_FALSE, B_TRUE,
1478                                    zstate);
1479                                eventstream_write(Z_EVT_ZONE_BOOTFAILED);
1480                        }
1481                        boot_args[0] = '\0';
1482                        break;
1483                case Z_HALT:
1484                        if (kernelcall) /* Invalid; can't happen */
```

```
1485                            abort();
1486                    if ((rval = zone_halt(zlogp, B_FALSE, B_FALSE, zstate))
1487                        != 0)
1488                            break;
1489                    eventstream_write(Z_EVT_ZONE_HALTED);
1490                    break;
1491            case Z_SHUTDOWN:
1492            case Z_REBOOT:
1493            case Z_NOTE_UNINSTALLING:
1494            case Z_MOUNT:
1495            case Z_UNMOUNT:
1496                    if (kernelcall) /* Invalid; can't happen */
1497                            abort();
1498                    zerror(zlogp, B_FALSE, "%s operation is invalid "
1499                        "for zones in state '%s'", z_cmd_name(cmd),
1500                        zone_state_str(zstate));
1501                    rval = -1;
1502                    break;
1503            }
1504            break;

1506    case ZONE_STATE_MOUNTED:
1507            switch (cmd) {
1508            case Z_UNMOUNT:
1509                    if (kernelcall) /* Invalid; can't happen */
1510                            abort();
1511                    rval = zone_halt(zlogp, B_TRUE, B_FALSE, zstate);
1512                    if (rval == 0) {
1513                            eventstream_write(Z_EVT_ZONE_HALTED);
1514                            (void) sema_post(&scratch_sem);
1515                    }
1516                    break;
1517            default:
1518                    if (kernelcall) /* Invalid; can't happen */
1519                            abort();
1520                    zerror(zlogp, B_FALSE, "%s operation is invalid "
1521                        "for zones in state '%s'", z_cmd_name(cmd),
1522                        zone_state_str(zstate));
1523                    rval = -1;
1524                    break;
1525            }
1526            break;

1528    case ZONE_STATE_RUNNING:
1529    case ZONE_STATE_SHUTTING_DOWN:
1530    case ZONE_STATE_DOWN:
1531            switch (cmd) {
1532            case Z_READY:
1533                    if ((rval = zone_halt(zlogp, B_FALSE, B_TRUE, zstate))
1534                        != 0)
1535                            break;
1536                    if ((rval = zone_ready(zlogp, Z_MNT_BOOT, zstate)) == 0)
1537                            eventstream_write(Z_EVT_ZONE_READIED);
1538                    else
1539                            eventstream_write(Z_EVT_ZONE_HALTED);
1540                    break;
1541            case Z_BOOT:
1542                    /*
1543                     * We could have two clients racing to boot this
1544                     * zone; the second client loses, but his request
1545                     * doesn't fail, since the zone is now in the desired
1546                     * state.
1547                     */
1548                    zerror(zlogp, B_FALSE, "zone is already booted");
1549                    rval = 0;
1550                    break;
```



```
1551            case Z_HALT:
1552                    if ((rval = zone_halt(zlogp, B_FALSE, B_FALSE, zstate))
1553                        != 0)
1554                            break;
1555                    eventstream_write(Z_EVT_ZONE_HALTED);
1556                    break;
1557            case Z_REBOOT:
1558                    (void) strlcpy(boot_args, zargp->bootbuf,
1559                        sizeof (boot_args));
1560                    eventstream_write(Z_EVT_ZONE_REBOOTING);
1561                    if ((rval = zone_halt(zlogp, B_FALSE, B_TRUE, zstate))
1562                        != 0) {
1563                            eventstream_write(Z_EVT_ZONE_BOOTFAILED);
1564                            boot_args[0] = '\0';
1565                            break;
1566                    }
1567                    if ((rval = zone_ready(zlogp, Z_MNT_BOOT, zstate))
1568                        != 0) {
1569                            eventstream_write(Z_EVT_ZONE_BOOTFAILED);
1570                            boot_args[0] = '\0';
1571                            break;
1572                    }
1573                    rval = zone_bootup(zlogp, zargp->bootbuf, zstate);
1574                    audit_put_record(zlogp, uc, rval, "reboot");
1575                    if (rval != 0) {
1576                            (void) zone_halt(zlogp, B_FALSE, B_TRUE,
1577                                zstate);
1578                            eventstream_write(Z_EVT_ZONE_BOOTFAILED);
1579                    }
1580                    boot_args[0] = '\0';
1581                    break;
1582            case Z_SHUTDOWN:
1583                    if ((rval = zone_graceful_shutdown(zlogp)) == 0) {
1584                            wait_shut = B_TRUE;
1585                    }
1586                    break;
1587            case Z_NOTE_UNINSTALLING:
1588            case Z_MOUNT:
1589            case Z_UNMOUNT:
1590                    zerror(zlogp, B_FALSE, "%s operation is invalid "
1591                        "for zones in state '%s'", z_cmd_name(cmd),
1592                        zone_state_str(zstate));
1593                    rval = -1;
1594                    break;
1595            }
1596            break;
1597    default:
1598            abort();
1599    }

1601    /*
1602     * Because the state of the zone may have changed, we make sure
1603     * to wake the console poller, which is in charge of initiating
1604     * the shutdown procedure as necessary.
1605     */
1606    eventstream_write(Z_EVT_NULL);

1608 out:
1609    (void) mutex_unlock(&lock);

1611    /* Wait for the Z_SHUTDOWN commands to complete */
1612    if (wait_shut)
1613            rval = zone_wait_shutdown(zlogp);

1615    if (kernelcall) {
1616            rvalp = NULL;
```

```
1617                    rlen = 0;
1618            } else {
1619                    rvalp->rval = rval;
1620            }
1621            if (uc != NULL)
1622                    ucred_free(uc);
1623            (void) door_return((char *)rvalp, rlen, NULL, 0);
1624            thr_exit(NULL);
1625 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    4948 Mon Feb 24 16:22:35 2014
new/usr/src/cmd/zoneadmd/zoneadmd.h
2594 implement graceful shutdown for local zones in zoneadm
**********************************************************
```

```
/*
 * 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.
 */

#ifndef _ZONEADMD_H
#define _ZONEADMD_H

#ifdef  __cplusplus
extern "C" {
#endif

#include <libdladm.h>

/*
 * Multi-threaded programs should avoid MT-unsafe library calls (i.e., any-
 * thing which could try to acquire a user-level lock unprotected by an atfork
 * handler) between fork(2) and exec(2).  See the pthread_atfork(3THR) man
 * page for details.  In particular, we want to avoid calls to zerror() in
 * such situations, as it calls setlocale(3c) which is susceptible to such
 * problems.  So instead we have the child use one of the special exit codes
 * below when needed, and the parent look out for such possibilities and call
 * zerror() there.
 *
 * Since 0, 1 and 2 are generally used for success, general error, and usage,
 * we start with 3.
 */
#define ZEXIT_FORK              3
#define ZEXIT_EXEC              4
#define ZEXIT_ZONE_ENTER        5

#define DEVFSADM        "devfsadm"
#define DEVFSADM_PATH   "/usr/sbin/devfsadm"

#define EXEC_PREFIX     "exec "
#define EXEC_LEN        (strlen(EXEC_PREFIX))

#define CLUSTER_BRAND_NAME      "cluster"
#define LABELED_BRAND_NAME      "labeled"
```



```
#define SHUTDOWN_WAIT           60
#define SHUTDOWN_DEFAULT        "/sbin/init 0"
#define SHUTDOWN_FMRI   "svc:/system/zones:default"

/* 0755 is the default directory mode. */
#define DEFAULT_DIR_MODE \
        (S_IRWXU | S_IRGRP | S_IXGRP | S_IROTH | S_IXOTH)
#define DEFAULT_DIR_USER -1     /* user ID for chown: -1 means don't change */
#define DEFAULT_DIR_GROUP -1    /* grp ID for chown: -1 means don't change */


typedef struct zlog {
        FILE *logfile;  /* file to log to */

        /*
         * The following are used if logging to a buffer.
         */
        char *log;      /* remaining log */
        size_t loglen;  /* size of remaining log */
        char *buf;      /* underlying storage */
        size_t buflen;  /* total len of 'buf' */
        char *locale;   /* locale to use for gettext() */
} zlog_t;
```

*unchanged_portion_omitted_*

```
**********************************************************
    4852 Mon Feb 24 16:22:35 2014
new/usr/src/lib/brand/ipkg/zone/config.xml
2594 implement graceful shutdown for local zones in zoneadm
**********************************************************
```

```
<?xml version="1.0"?>

<!--
 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) 2006, 2010, Oracle and/or its affiliates. All rights reserved.

 DO NOT EDIT THIS FILE.
 Copyright 2014 Nexenta Systems, Inc. All rights reserved.
-->

<!DOCTYPE brand PUBLIC "-//Sun Microsystems Inc//DTD Brands//EN"
    "file:///usr/share/lib/xml/dtd/brand.dtd.1">

<brand name="ipkg">
        <modname></modname>

        <initname>/sbin/init</initname>
        <login_cmd>/usr/bin/login -z %Z %u</login_cmd>
        <forcedlogin_cmd>/usr/bin/login -z %Z -f %u</forcedlogin_cmd>
        <user_cmd>/usr/bin/getent passwd %u</user_cmd>

        <!-- We may not be able to do the create in pkg(1) proper. -->
        <install>/usr/lib/brand/ipkg/pkgcreatezone -z %z -R %R</install>
        <installopts>a:c:d:e:hk:P:p:suv</installopts>
        <boot></boot>
        <sysboot>/usr/lib/brand/ipkg/prestate %z %R 2 0</sysboot>
        <halt></halt>
        <shutdown>/usr/sbin/shutdown -y -g0 -i5</shutdown>
        <verify_cfg></verify_cfg>
        <verify_adm></verify_adm>
        <postclone></postclone>
        <postinstall></postinstall>
        <attach>/usr/lib/brand/ipkg/attach %z %R</attach>
        <detach>/usr/lib/brand/ipkg/detach -z %z -R %R</detach>
        <clone>/usr/lib/brand/ipkg/clone -z %z -R %R</clone>
        <uninstall>/usr/lib/brand/ipkg/uninstall %z %R</uninstall>
        <prestatechange>/usr/lib/brand/ipkg/prestate %z %R</prestatechange>
        <poststatechange>/usr/lib/brand/ipkg/poststate %z %R</poststatechange>
        <query>/usr/lib/brand/shared/query %z %R</query>

        <privilege set="default" name="contract_event" />
        <privilege set="default" name="contract_identity" />
        <privilege set="default" name="contract_observer" />
```



```
        <privilege set="default" name="file_chown" />
        <privilege set="default" name="file_chown_self" />
        <privilege set="default" name="file_dac_execute" />
        <privilege set="default" name="file_dac_read" />
        <privilege set="default" name="file_dac_search" />
        <privilege set="default" name="file_dac_write" />
        <privilege set="default" name="file_owner" />
        <privilege set="default" name="file_setid" />
        <privilege set="default" name="ipc_dac_read" />
        <privilege set="default" name="ipc_dac_write" />
        <privilege set="default" name="ipc_owner" />
        <privilege set="default" name="net_bindmlp" />
        <privilege set="default" name="net_icmpaccess" />
        <privilege set="default" name="net_mac_aware" />
        <privilege set="default" name="net_observability" />
        <privilege set="default" name="net_privaddr" />
        <privilege set="default" name="net_rawaccess" ip-type="exclusive" />
        <privilege set="default" name="proc_chroot" />
        <privilege set="default" name="sys_audit" />
        <privilege set="default" name="proc_audit" />
        <privilege set="default" name="proc_lock_memory" />
        <privilege set="default" name="proc_owner" />
        <privilege set="default" name="proc_setid" />
        <privilege set="default" name="proc_taskid" />
        <privilege set="default" name="sys_acct" />
        <privilege set="default" name="sys_admin" />
        <privilege set="default" name="sys_ip_config" ip-type="exclusive" />
        <privilege set="default" name="sys_iptun_config" ip-type="exclusive" />
        <privilege set="default" name="sys_mount" />
        <privilege set="default" name="sys_nfs" />
        <privilege set="default" name="sys_resource" />
        <privilege set="default" name="sys_ppp_config" ip-type="exclusive" />

        <privilege set="prohibited" name="dtrace_kernel" />
        <privilege set="prohibited" name="proc_zone" />
        <privilege set="prohibited" name="sys_config" />
        <privilege set="prohibited" name="sys_devices" />
        <privilege set="prohibited" name="sys_ip_config" ip-type="shared" />
        <privilege set="prohibited" name="sys_linkdir" />
        <privilege set="prohibited" name="sys_net_config" />
        <privilege set="prohibited" name="sys_res_config" />
        <privilege set="prohibited" name="sys_suser_compat" />
        <privilege set="prohibited" name="xvm_control" />
        <privilege set="prohibited" name="virt_manage" />
        <privilege set="prohibited" name="sys_ppp_config" ip-type="shared" />

        <privilege set="required" name="proc_exec" />
        <privilege set="required" name="proc_fork" />
        <privilege set="required" name="sys_ip_config" ip-type="exclusive" />
        <privilege set="required" name="sys_mount" />
</brand>
```

```
***********************************************************
    4856 Mon Feb 24 16:22:36 2014
new/usr/src/lib/brand/labeled/zone/config.xml
2594 implement graceful shutdown for local zones in zoneadm
***********************************************************
```

```
<?xml version="1.0"?>

<!--
 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) 2006, 2010, Oracle and/or its affiliates. All rights reserved.

 DO NOT EDIT THIS FILE.
 Copyright 2014 Nexenta Systems, Inc. All rights reserved.
-->

<!DOCTYPE brand PUBLIC "-//Sun Microsystems Inc//DTD Brands//EN"
    "file:///usr/share/lib/xml/dtd/brand.dtd.1">

<brand name="labeled">
        <modname></modname>

        <initname>/sbin/init</initname>
        <login_cmd>/usr/bin/login -z %Z %u</login_cmd>
        <forcedlogin_cmd>/usr/bin/login -z %Z -f %u</forcedlogin_cmd>

        <user_cmd>/usr/bin/getent passwd %u</user_cmd>

        <!-- We may not be able to do the create in pkg(1) proper. -->
        <install>/usr/lib/brand/ipkg/pkgcreatezone -z %z -R %R</install>
        <installopts>a:c:d:e:hk:P:p:suv</installopts>
        <boot></boot>
        <sysboot>/usr/lib/brand/ipkg/prestate %z %R 2 0</sysboot>
        <halt></halt>
        <shutdown>/usr/sbin/shutdown -y -g0 -i5</shutdown>
        <verify_cfg></verify_cfg>
        <verify_adm></verify_adm>
        <postclone></postclone>
        <postinstall></postinstall>
        <attach>/usr/lib/brand/ipkg/attach %z %R</attach>
        <detach>/usr/lib/brand/ipkg/detach -z %z -R %R</detach>
        <clone>/usr/lib/brand/ipkg/clone -z %z -R %R</clone>
        <uninstall>/usr/lib/brand/ipkg/uninstall %z %R</uninstall>
        <prestatechange>/usr/lib/brand/ipkg/prestate %z %R</prestatechange>
        <poststatechange>/usr/lib/brand/ipkg/poststate %z %R</poststatechange>
        <query>/usr/lib/brand/shared/query %z %R</query>

        <privilege set="default" name="contract_event" />
        <privilege set="default" name="contract_identity" />
```



```
        <privilege set="default" name="contract_observer" />
        <privilege set="default" name="file_chown" />
        <privilege set="default" name="file_chown_self" />
        <privilege set="default" name="file_dac_execute" />
        <privilege set="default" name="file_dac_read" />
        <privilege set="default" name="file_dac_search" />
        <privilege set="default" name="file_dac_write" />
        <privilege set="default" name="file_owner" />
        <privilege set="default" name="file_setid" />
        <privilege set="default" name="ipc_dac_read" />
        <privilege set="default" name="ipc_dac_write" />
        <privilege set="default" name="ipc_owner" />
        <privilege set="default" name="net_bindmlp" />
        <privilege set="default" name="net_icmpaccess" />
        <privilege set="default" name="net_mac_aware" />
        <privilege set="default" name="net_observability" />
        <privilege set="default" name="net_privaddr" />
        <privilege set="default" name="net_rawaccess" ip-type="exclusive" />
        <privilege set="default" name="proc_chroot" />
        <privilege set="default" name="sys_audit" />
        <privilege set="default" name="proc_audit" />
        <privilege set="default" name="proc_lock_memory" />
        <privilege set="default" name="proc_owner" />
        <privilege set="default" name="proc_setid" />
        <privilege set="default" name="proc_taskid" />
        <privilege set="default" name="sys_acct" />
        <privilege set="default" name="sys_admin" />
        <privilege set="default" name="sys_ip_config" ip-type="exclusive" />
        <privilege set="default" name="sys_iptun_config" ip-type="exclusive" />
        <privilege set="default" name="sys_mount" />
        <privilege set="default" name="sys_nfs" />
        <privilege set="default" name="sys_resource" />
        <privilege set="default" name="sys_ppp_config" ip-type="exclusive" />

        <privilege set="prohibited" name="dtrace_kernel" />
        <privilege set="prohibited" name="proc_zone" />
        <privilege set="prohibited" name="sys_config" />
        <privilege set="prohibited" name="sys_devices" />
        <privilege set="prohibited" name="sys_ip_config" ip-type="shared" />
        <privilege set="prohibited" name="sys_linkdir" />
        <privilege set="prohibited" name="sys_net_config" />
        <privilege set="prohibited" name="sys_res_config" />
        <privilege set="prohibited" name="sys_suser_compat" />
        <privilege set="prohibited" name="xvm_control" />
        <privilege set="prohibited" name="virt_manage" />
        <privilege set="prohibited" name="sys_ppp_config" ip-type="shared" />

        <privilege set="required" name="proc_exec" />
        <privilege set="required" name="proc_fork" />
        <privilege set="required" name="sys_ip_config" ip-type="exclusive" />
        <privilege set="required" name="sys_mount" />
</brand>
```

```
**********************************************************
    4990 Mon Feb 24 16:22:36 2014
new/usr/src/lib/brand/solaris10/zone/config.xml
2594 implement graceful shutdown for local zones in zoneadm
**********************************************************
```

```
<?xml version="1.0"?>

<!--
 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) 2009, 2010, Oracle and/or its affiliates. All rights reserved.

 DO NOT EDIT THIS FILE.
 Copyright 2014 Nexenta Systems, Inc. All rights reserved.
-->

<!DOCTYPE brand PUBLIC "-//Sun Microsystems Inc//DTD Brands//EN"
    "file:///usr/share/lib/xml/dtd/brand.dtd.1">

<brand name="solaris10">
        <modname>s10_brand</modname>
        <initname>/sbin/init</initname>
        <login_cmd>/usr/bin/login -z %Z %u</login_cmd>
        <forcedlogin_cmd>/usr/bin/login -z %Z -f %u</forcedlogin_cmd>
        <user_cmd>/usr/bin/getent passwd %u</user_cmd>

        <install>/usr/lib/brand/solaris10/image_install %z %R</install>
        <installopts>a:d:Fpsuv</installopts>
        <boot>/usr/lib/brand/solaris10/s10_boot %z %R</boot>
        <sysboot>/usr/lib/brand/solaris10/prestate %z %R 2 0</sysboot>
        <halt></halt>
        <shutdown>/usr/sbin/shutdown -y -g0 -i5</shutdown>
        <verify_cfg>/usr/lib/brand/solaris10/s10_support verify</verify_cfg>
        <verify_adm></verify_adm>
        <postattach>/usr/lib/brand/solaris10/postattach %z %R</postattach>
        <attach>/usr/lib/brand/solaris10/attach %z %R</attach>
        <detach>/usr/lib/brand/solaris10/detach %z %R</detach>
        <clone>/usr/lib/brand/solaris10/clone -z %z -R %R</clone>
        <preuninstall>/usr/lib/brand/solaris10/preuninstall %z %R</preuninstall>
        <uninstall>/usr/lib/brand/solaris10/uninstall %z %R</uninstall>
        <prestatechange>/usr/lib/brand/solaris10/prestate %z %R</prestatechange>
        <poststatechange>/usr/lib/brand/solaris10/poststate %z %R</poststatechan
        <query>/usr/lib/brand/shared/query %z %R</query>

        <privilege set="default" name="contract_event" />
        <privilege set="default" name="contract_identity" />
        <privilege set="default" name="contract_observer" />
        <privilege set="default" name="file_chown" />
        <privilege set="default" name="file_chown_self" />
```



```
        <privilege set="default" name="file_dac_execute" />
        <privilege set="default" name="file_dac_read" />
        <privilege set="default" name="file_dac_search" />
        <privilege set="default" name="file_dac_write" />
        <privilege set="default" name="file_owner" />
        <privilege set="default" name="file_setid" />
        <privilege set="default" name="ipc_dac_read" />
        <privilege set="default" name="ipc_dac_write" />
        <privilege set="default" name="ipc_owner" />
        <privilege set="default" name="net_bindmlp" />
        <privilege set="default" name="net_icmpaccess" />
        <privilege set="default" name="net_mac_aware" />
        <privilege set="default" name="net_observability" />
        <privilege set="default" name="net_privaddr" />
        <privilege set="default" name="net_rawaccess" ip-type="exclusive" />
        <privilege set="default" name="proc_chroot" />
        <privilege set="default" name="sys_audit" />
        <privilege set="default" name="proc_audit" />
        <privilege set="default" name="proc_lock_memory" />
        <privilege set="default" name="proc_owner" />
        <privilege set="default" name="proc_setid" />
        <privilege set="default" name="proc_taskid" />
        <privilege set="default" name="sys_acct" />
        <privilege set="default" name="sys_admin" />
        <privilege set="default" name="sys_ip_config" ip-type="exclusive" />
        <privilege set="default" name="sys_iptun_config" ip-type="exclusive" />
        <privilege set="default" name="sys_mount" />
        <privilege set="default" name="sys_nfs" />
        <privilege set="default" name="sys_resource" />
        <privilege set="default" name="sys_ppp_config" ip-type="exclusive" />

        <privilege set="prohibited" name="dtrace_kernel" />
        <privilege set="prohibited" name="proc_zone" />
        <privilege set="prohibited" name="sys_config" />
        <privilege set="prohibited" name="sys_devices" />
        <privilege set="prohibited" name="sys_ip_config" ip-type="shared" />
        <privilege set="prohibited" name="sys_linkdir" />
        <privilege set="prohibited" name="sys_net_config" />
        <privilege set="prohibited" name="sys_res_config" />
        <privilege set="prohibited" name="sys_suser_compat" />
        <privilege set="prohibited" name="xvm_control" />
        <privilege set="prohibited" name="virt_manage" />
        <privilege set="prohibited" name="sys_ppp_config" ip-type="shared" />

        <privilege set="required" name="proc_exec" />
        <privilege set="required" name="proc_fork" />
        <privilege set="required" name="sys_ip_config" ip-type="exclusive" />
        <privilege set="required" name="sys_mount" />
</brand>
```

```
**********************************************************
   27482 Mon Feb 24 16:22:36 2014
new/usr/src/lib/libbrand/common/libbrand.c
2594 implement graceful shutdown for local zones in zoneadm
**********************************************************
```

```
/*
 * 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) 2006, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2014 Nexenta Systems, Inc. All rights reserved.
 */

#include <assert.h>
#include <dirent.h>
#include <errno.h>
#include <fnmatch.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include <strings.h>
#include <synch.h>
#include <sys/brand.h>
#include <sys/fcntl.h>
#include <sys/param.h>
#include <sys/stat.h>
#include <sys/systeminfo.h>
#include <sys/types.h>
#include <thread.h>
#include <zone.h>

#include <libbrand_impl.h>
#include <libbrand.h>

#define DTD_ELEM_ATTACH         ((const xmlChar *) "attach")
#define DTD_ELEM_BOOT           ((const xmlChar *) "boot")
#define DTD_ELEM_BRAND          ((const xmlChar *) "brand")
#define DTD_ELEM_CLONE          ((const xmlChar *) "clone")
#define DTD_ELEM_COMMENT        ((const xmlChar *) "comment")
#define DTD_ELEM_DETACH         ((const xmlChar *) "detach")
#define DTD_ELEM_DEVICE         ((const xmlChar *) "device")
#define DTD_ELEM_GLOBAL_MOUNT   ((const xmlChar *) "global_mount")
#define DTD_ELEM_HALT           ((const xmlChar *) "halt")
#define DTD_ELEM_INITNAME       ((const xmlChar *) "initname")
#define DTD_ELEM_INSTALL        ((const xmlChar *) "install")
#define DTD_ELEM_INSTALLOPTS    ((const xmlChar *) "installopts")
#define DTD_ELEM_LOGIN_CMD      ((const xmlChar *) "login_cmd")
#define DTD_ELEM_FORCELOGIN_CMD ((const xmlChar *) "forcedlogin_cmd")
```



```
#define DTD_ELEM_MODNAME        ((const xmlChar *) "modname")
#define DTD_ELEM_MOUNT          ((const xmlChar *) "mount")
#define DTD_ELEM_POSTATTACH     ((const xmlChar *) "postattach")
#define DTD_ELEM_POSTCLONE      ((const xmlChar *) "postclone")
#define DTD_ELEM_POSTINSTALL    ((const xmlChar *) "postinstall")
#define DTD_ELEM_POSTSNAP       ((const xmlChar *) "postsnap")
#define DTD_ELEM_POSTSTATECHG   ((const xmlChar *) "poststatechange")
#define DTD_ELEM_PREDETACH      ((const xmlChar *) "predetach")
#define DTD_ELEM_PRESNAP        ((const xmlChar *) "presnap")
#define DTD_ELEM_PRESTATECHG    ((const xmlChar *) "prestatechange")
#define DTD_ELEM_PREUNINSTALL   ((const xmlChar *) "preuninstall")
#define DTD_ELEM_PRIVILEGE      ((const xmlChar *) "privilege")
#define DTD_ELEM_QUERY          ((const xmlChar *) "query")
#define DTD_ELEM_SHUTDOWN       ((const xmlChar *) "shutdown")
#define DTD_ELEM_SYMLINK        ((const xmlChar *) "symlink")
#define DTD_ELEM_SYSBOOT        ((const xmlChar *) "sysboot")
#define DTD_ELEM_UNINSTALL      ((const xmlChar *) "uninstall")
#define DTD_ELEM_USER_CMD       ((const xmlChar *) "user_cmd")
#define DTD_ELEM_VALIDSNAP      ((const xmlChar *) "validatesnap")
#define DTD_ELEM_VERIFY_CFG     ((const xmlChar *) "verify_cfg")
#define DTD_ELEM_VERIFY_ADM     ((const xmlChar *) "verify_adm")

#define DTD_ATTR_ALLOWEXCL      ((const xmlChar *) "allow-exclusive-ip")
#define DTD_ATTR_ARCH           ((const xmlChar *) "arch")
#define DTD_ATTR_DIRECTORY      ((const xmlChar *) "directory")
#define DTD_ATTR_IPTYPE         ((const xmlChar *) "ip-type")
#define DTD_ATTR_MATCH          ((const xmlChar *) "match")
#define DTD_ATTR_MODE           ((const xmlChar *) "mode")
#define DTD_ATTR_NAME           ((const xmlChar *) "name")
#define DTD_ATTR_OPT            ((const xmlChar *) "opt")
#define DTD_ATTR_PATH           ((const xmlChar *) "path")
#define DTD_ATTR_SET            ((const xmlChar *) "set")
#define DTD_ATTR_SOURCE         ((const xmlChar *) "source")
#define DTD_ATTR_SPECIAL        ((const xmlChar *) "special")
#define DTD_ATTR_TARGET         ((const xmlChar *) "target")
#define DTD_ATTR_TYPE           ((const xmlChar *) "type")

#define DTD_ENTITY_TRUE         "true"

static volatile boolean_t       libbrand_initialized = B_FALSE;
static char                     i_curr_arch[MAXNAMELEN];
static char                     i_curr_zone[ZONENAME_MAX];

/*ARGSUSED*/
static void
brand_error_func(void *ctx, const char *msg, ...)
{
        /*
         * Ignore error messages from libxml
         */
}
```

*_____unchanged_portion_omitted_*

```
int
brand_get_shutdown(brand_handle_t bh, const char *zonename,
    const char *zonepath, char *buf, size_t len)
{
        struct brand_handle *bhp = (struct brand_handle *)bh;
        return (brand_get_value(bhp, zonename, zonepath, NULL, NULL,
            buf, len, DTD_ELEM_SHUTDOWN, B_TRUE, B_TRUE));
}

int
brand_get_initname(brand_handle_t bh, char *buf, size_t len)
{
        struct brand_handle *bhp = (struct brand_handle *)bh;
```

```
 519         return (brand_get_value(bhp, NULL, NULL, NULL, NULL,
 520             buf, len, DTD_ELEM_INITNAME, B_FALSE, B_FALSE));
 521 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    4646 Mon Feb 24 16:22:36 2014
new/usr/src/lib/libbrand/common/libbrand.h
2594 implement graceful shutdown for local zones in zoneadm
**********************************************************
```

```
/*
 * 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) 2006, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2014 Nexenta Systems, Inc. All rights reserved.
 */

#ifndef _LIBBRAND_H
#define _LIBBRAND_H

#ifdef  __cplusplus
extern "C" {
#endif

#include <sys/types.h>

typedef struct __brand_handle *brand_handle_t;

typedef struct priv_iter_s {
        char    *pi_name;
        char    *pi_set;
        char    *pi_iptype;
} priv_iter_t;

extern brand_handle_t brand_open(const char *);
extern void brand_close(brand_handle_t);

extern boolean_t brand_allow_exclusive_ip(brand_handle_t);

extern int brand_get_attach(brand_handle_t, const char *, const char *,
    char *, size_t);
extern int brand_get_boot(brand_handle_t, const char *, const char *,
    char *, size_t);
extern int brand_get_brandname(brand_handle_t, char *, size_t);
extern int brand_get_clone(brand_handle_t, const char *, const char *,
    char *, size_t);
extern int brand_get_detach(brand_handle_t, const char *, const char *,
    char *, size_t);
extern int brand_get_shutdown(brand_handle_t, const char *, const char *,
    char *, size_t);
extern int brand_get_halt(brand_handle_t, const char *, const char *,
    char *, size_t);
```



```
extern int brand_get_initname(brand_handle_t, char *, size_t);
extern int brand_get_install(brand_handle_t, const char *, const char *,
    char *, size_t);
extern int brand_get_installopts(brand_handle_t, char *, size_t);
extern int brand_get_login_cmd(brand_handle_t, const char *, char *, size_t);
extern int brand_get_forcedlogin_cmd(brand_handle_t, const char *,
    char *, size_t);
extern int brand_get_modname(brand_handle_t, char *, size_t);
extern int brand_get_postattach(brand_handle_t, const char *, const char *,
    char *, size_t);
extern int brand_get_postclone(brand_handle_t, const char *, const char *,
    char *, size_t);
extern int brand_get_postinstall(brand_handle_t, const char *, const char *,
    char *, size_t);
extern int brand_get_postsnap(brand_handle_t, const char *, const char *,
    char *, size_t);
extern int brand_get_poststatechange(brand_handle_t, const char *, const char *,
    char *, size_t);
extern int brand_get_predetach(brand_handle_t, const char *, const char *,
    char *, size_t);
extern int brand_get_presnap(brand_handle_t, const char *, const char *,
    char *, size_t);
extern int brand_get_prestatechange(brand_handle_t, const char *, const char *,
    char *, size_t);
extern int brand_get_preuninstall(brand_handle_t, const char *, const char *,
    char *, size_t);
extern int brand_get_query(brand_handle_t, const char *, const char *,
    char *, size_t);
extern int brand_get_uninstall(brand_handle_t, const char *, const char *,
    char *, size_t);
extern int brand_get_validatesnap(brand_handle_t, const char *, const char *,
    char *, size_t);
extern int brand_get_user_cmd(brand_handle_t, const char *, char *, size_t);
extern int brand_get_verify_cfg(brand_handle_t, char *, size_t);
extern int brand_get_verify_adm(brand_handle_t, const char *, const char *,
    char *, size_t);
extern int brand_get_sysboot(brand_handle_t, const char *, const char *, char *,
    size_t);

extern int brand_config_iter_privilege(brand_handle_t,
    int (*func)(void *, priv_iter_t *), void *);

extern int brand_platform_iter_devices(brand_handle_t, const char *,
    int (*)(void *, const char *, const char *), void *, const char *);
extern int brand_platform_iter_gmounts(brand_handle_t, const char *,
    int (*)(void *, const char *, const char *, const char *, const char *),
    void *);
extern int brand_platform_iter_link(brand_handle_t, int (*)(void *,
    const char *, const char *), void *);
extern int brand_platform_iter_mounts(brand_handle_t, int (*)(void *,
    const char *, const char *, const char *, const char *), void *);

#ifdef  __cplusplus
}
_____unchanged_portion_omitted_
```

```
**********************************************************
    2203 Mon Feb 24 16:22:36 2014
new/usr/src/lib/libbrand/common/mapfile-vers
2594 implement graceful shutdown for local zones in zoneadm
**********************************************************
#
# 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) 2006, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright 2014 Nexenta Systems, Inc. All rights reserved.
#

#
# MAPFILE HEADER START
#
# WARNING:  STOP NOW.  DO NOT MODIFY THIS FILE.
# Object versioning must comply with the rules detailed in
#
#       usr/src/lib/README.mapfiles
#
# You should not be making modifications here until you've read the most current
# copy of that file. If you need help, contact a gatekeeper for guidance.
#
# MAPFILE HEADER END
#

$mapfile_version 2

SYMBOL_VERSION SUNWprivate {
    global:
        brand_allow_exclusive_ip;
        brand_close;
        brand_config_iter_privilege;
        brand_get_attach;
        brand_get_boot;
        brand_get_brandname;
        brand_get_clone;
        brand_get_detach;
        brand_get_forcedlogin_cmd;
        brand_get_halt;
        brand_get_initname;
        brand_get_install;
        brand_get_installopts;
        brand_get_login_cmd;
        brand_get_modname;
        brand_get_postattach;
        brand_get_postclone;
        brand_get_postinstall;
```



```
        brand_get_postsnap;
        brand_get_poststatechange;
        brand_get_predetach;
        brand_get_presnap;
        brand_get_prestatechange;
        brand_get_preuninstall;
        brand_get_query;
        brand_get_sysboot;
        brand_get_shutdown;
        brand_get_uninstall;
        brand_get_user_cmd;
        brand_get_validatesnap;
        brand_get_verify_adm;
        brand_get_verify_cfg;
        brand_open;
        brand_platform_iter_devices;
        brand_platform_iter_gmounts;
        brand_platform_iter_link;
        brand_platform_iter_mounts;
    local:
        *;
};
______unchanged_portion_omitted_
```

```
**********************************************************
   19242 Mon Feb 24 16:22:37 2014
new/usr/src/lib/libbrand/dtd/brand.dtd.1
2594 implement graceful shutdown for local zones in zoneadm
**********************************************************
```

```
<?xml version='1.0' encoding='UTF-8' ?>

<!--
 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) 2006, 2010, Oracle and/or its affiliates. All rights reserved.

 DO NOT EDIT THIS FILE.

 Copyright 2014 Nexenta Systems, Inc. All rights reserved.
-->

<!--
  verify_cfg

    Identifies the program to be invoked by zonecfg to verify that the
    zone's configuration is legal, and that all the configured devices,
    attributes, etc. are legal for this brand.

    The program is called with a single argument: the path to a file
    containing a temporary config.xml file the zone.  It should return 0
    on success and non-0 on failure.  Any detailed error messages should be
    displayed to stderr.

    It has no attributes.

-->
<!ELEMENT verify_cfg    (#PCDATA) >
<!ATTLIST verify_cfg>
<!--
  verify_adm

    Identifies the program invoked by zoneadm to perform brand-specific
    checks as to the viability of a zone on this specific machine.

    The following replacements are performed:

      %z        Name of zone
      %R        Zonepath of zone
      Additional arguments, if any, are appended.

    The program should return 0 on success and non-0 on failure.  Any
    detailed error messages should be displayed to stderr.
```



```
    It has no attributes.

-->
<!ELEMENT verify_adm    (#PCDATA) >
<!ATTLIST verify_adm>

<!--
  install

    Identifies the program to invoke when installing a zone.  The following
    replacements are performed:

      %z        Name of zone
      %R        Zonepath of zone
      Additional arguments, if any, are appended.

    It has no attributes.
-->
<!ELEMENT install       (#PCDATA) >
<!ATTLIST install>

<!--
  installopts

    Identifies the command-line options supported by the brand's
    installation program, allowing zoneadm to parse the install line
    properly.

    It has no attributes.
-->
<!ELEMENT installopts   (#PCDATA) >
<!ATTLIST installopts>

<!--
  boot

    This is a program which gets run by zoneadmd when a zone is booted.
    The program will be invoked as the last step in the zone booting
    process before the the first process is spawned inside the zone.

    If this programs succeeds it should not generate any output.
    If this program returns an error, any output generated by the
    program will be sent to the zoneadmd message log.

    The following replacements are performed:

      %z        Name of zone
      %R        Zonepath of zone
      Additional arguments, if any, are appended.

    It has no attributes.
-->
<!ELEMENT boot  (#PCDATA) >
<!ATTLIST boot>

<!--
  sysboot

    This is a program that will be run by zoneadm during system boot for an
    installed zone that won't automatically boot.

    If the program succeeds, then it should not generate output.
    If the program returns an error, then the output it generates will be
    sent to the zones SMF service's message log.

    The following replacements are performed:
```

```
129       %z        Name of the target zone
130       %R        Zonepath of the target zone
131       Additional arguments, if any, are appended.

133     This element has no attributes.
134 -->
135 <!ELEMENT sysboot       (#PCDATA) >
136 <!ATTLIST sysboot>

138 <!--
139   halt

141     This is a program which gets run by zoneadmd when a zone is being
142     halted.  This callback is provided to allow a brand to cleanup any
143     special configuration that was setup during boot.

145     This program will also be invoked by zoneadmd if any part of the zone
146     booting process fail, even if the booting process failed before the
147     brand boot program was invoked.  It is also possible that if the zone
148     fails to halt after invoking this program, future attempts to halt the
149     zone will invoke this program again.  So this program should be
150     designed to clean up any resources allocated to a zone but it should
151     also be able to gracefully handle the case where resources that it
152     expects to release are not actually allocated (or have been already
153     released.)

155     If this programs succeeds it should not generate any output.  If this
156     program returns an error, any output generated by the program will be
157     sent to the zoneadmd message log.

159     The following replacements are performed:

161       %z        Name of zone
162       %R        Zonepath of zone
163       Additional arguments, if any, are appended.

165     It has no attributes.
166 -->
167 <!ELEMENT halt  (#PCDATA) >
168 <!ATTLIST halt>

170 <!--
171   shutdown

173     This is a program which gets run by zoneadmd when a zone is being
174     shutdown gracefully. Currently only asynchronous mode is supported.

176     If this program succeeds it should not generate any output. If this
177     program returns an error, any output generated by the program will be
178     sent to the zoneadmd message log.

180     The following replacements are performed:

182       %z        Name of zone
183       %R        Zonepath of zone
184       Additional arguments, if any, are appended.

186     It has no attributes.
187 -->
188 <!ELEMENT shutdown      (#PCDATA) >
189 <!ATTLIST shutdown>

191 <!--
192   modname
```



```
194     Path to the kernel module that implements the kernel-level
195     functionality of the brand.

197     It has no attributes.
198 -->
199 <!ELEMENT modname       (#PCDATA) >
200 <!ATTLIST modname>

202 <!--
203   initname

205     Path to the initial executable that should be launched when booting a
206     branded zone.

208     It has no attributes.
209 -->
210 <!ELEMENT initname      (#PCDATA) >
211 <!ATTLIST initname>

213 <!--
214   login_cmd

216     Path to the initial login binary that should be executed when
217     attempting to zlogin into a branded zone.

219     The following replacements are performed:

221       %Z        Name of the current zone
222       %u        User login name

224     It has no attributes.
225 -->
226 <!ELEMENT login_cmd     (#PCDATA) >
227 <!ATTLIST login_cmd>

229 <!--
230   forcedlogin_cmd

232     Path to the initial login binary that should be executed when
233     attempting to zlogin into a branded zone without authentication.

235     The following replacements are performed:

237       %Z        Name of the current zone
238       %u        User login name

240     It has no attributes.
241 -->
242 <!ELEMENT forcedlogin_cmd       (#PCDATA) >
243 <!ATTLIST forcedlogin_cmd>

245 <!--
246   user_cmd

248     Path to the binary that will translate a user name to a passwd(4) entry.

250     The following replacements are performed:

252       %u        User login name

254     It has no attributes.  The passwd(4) entry is used to determine $LOGNAME,
255     $HOME, and $SHELL for non-interactive "zlogin -l <user> <cmd>".
256 -->
257 <!ELEMENT user_cmd   (#PCDATA) >
258 <!ATTLIST user_cmd>
```

```
<!--
  attach

    Path to a hook that will perform any necessary processing on
    a zone to allow it to be attached.  The zone will be in the "configured"
    state when this hook is run.  This hook is never called when the zone
    is "force attached" (-F).

    If this hook exits with a non-zero exit status, the attach operation
    will fail.

    The following replacements are performed:

      %z        Name of zone
      %R        Zonepath of zone
      Additional arguments, if any, are appended.

    If no hook is provided, the internal zoneadm attach code will be used.

    It has no attributes.
-->
<!ELEMENT attach        (#PCDATA) >
<!ATTLIST attach>

<!--
  postattach

    Path to a hook that will perform any necessary post-processing on
    a zone after it has been attached.  The zone will be in the "installed"
    state when this hook is run.  This hook is never called when the zone
    is "force attached" (-F).

    If this hook exits with a non-zero exit status, the attach operation
    will fail and the zone state will be reset to "configured".

    The following replacements are performed:

      %z        Name of zone
      %R        Zonepath of zone
      Additional arguments, if any, are appended.

    It has no attributes.
-->
<!ELEMENT postattach    (#PCDATA) >
<!ATTLIST postattach>

<!--
  postclone

    Path to a hook that will perform any necessary post-processing on
    a zone after it has been cloned.  The zone will be in the "incomplete"
    state when this hook is run.

    If this hook exits with a non-zero exit status, the clone operation
    will fail and the zone will be left in the "incomplete" state,
    otherwise the state will be changed to the "installed" state.

    The following replacements are performed:

      %z        Name of zone
      %R        Zonepath of zone
      Additional arguments, if any, are appended.

    It has no attributes.
-->
<!ELEMENT postclone     (#PCDATA) >
```



```
<!ATTLIST postclone>

<!--
  postinstall

    Path to a script that will perform any necessary post-processing on
    a zone after it has been freshly installed.  This hook will run after the
    install hook completes and the zone is in the installed state.  The
    additional arguments are the same as what is passed to the install hook.

    The following replacements are performed:

      %z        Name of zone
      %R        Zonepath of zone
      Additional arguments, if any, are appended.

    It has no attributes.
-->
<!ELEMENT postinstall   (#PCDATA) >
<!ATTLIST postinstall>

<!--
  predetach

    Path to a hook that will perform any necessary pre-processing on
    a zone before it is detached.  The zone will be in the "installed"
    state when this hook is run.

    It is possible that if the zone fails to detach after invoking this
    hook, future attempts to detach the zone will invoke this hook again.
    So this hook should be designed to gracefully handle the case where
    it is run multiple times on the same zone.  If this hook exits with
    a non-zero exit status, the detach operation will fail.

    This hook is most commonly used when there is pre-processing for detaching
    a zone but the built-in detach support will be used for the actual
    detach.  Otherwise, if a detach hook is provided, then it can be used
    to do both preprocessing as well as the actual detach.

    The following replacements are performed:

      %z        Name of zone
      %R        Zonepath of zone
      Additional arguments, if any, are appended.

    It has no attributes.
-->
<!ELEMENT predetach     (#PCDATA) >
<!ATTLIST predetach>

<!--
  detach

    Path to a hook that will perform any necessary processing on
    a zone to allow it to be detached.  The zone will be in the "installed"
    state when this hook is run.

    It is possible that if the zone fails to detach while running this
    hook, future attempts to detach the zone will invoke this hook again.
    So this hook should be designed to gracefully handle the case where
    it is run multiple times on the same zone.  If this hook exits with
    a non-zero exit status, the detach operation will fail and the zone will
    be left in the "installed" state, otherwise the state will be changed
    to "configured".

    The following replacements are performed:
```

```
393       %z        Name of zone
394       %R        Zonepath of zone
395       Additional arguments, if any, are appended.

397     If no hook is provided, the internal zoneadm detach code will be used.

399     It has no attributes.
400 -->
401 <!ELEMENT detach        (#PCDATA) >
402 <!ATTLIST detach>

404 <!--
405   clone
406     Path to a hook that will perform any necessary processing on a zone to
407     allow it to be installed via cloning.  Cloning is an alternative to
408     installing so this hook should result in the same effect for the zone.
409     The zone will be in the "incomplete" state when this hook is run.

411     If this hook exits with a non-zero exit status, the clone operation
412     will fail and the zone will be left in the "incomplete" state, otherwise
413     the state will be changed to "installed".

415     The following replacements are performed:

417       %z        Name of zone
418       %R        Zonepath of zone
419       1st arg   name of source zone
420       Additional arguments, if any, are appended.

422     If no hook is provided, the internal zoneadm cloning code will be used.
423 -->
424 <!ELEMENT clone (#PCDATA) >
425 <!ATTLIST clone>

427 <!--
428   preuninstall

430     Path to a script that will perform any necessary pre-processing on
431     a zone before it is uninstalled.  The zone will be in the "installed"
432     state when this hook is run.

434     It is possible that if the zone fails to uninstall after invoking this
435     hook, future attempts to uninstall the zone will invoke this hook
436     again.  So this hook should be designed to gracefully handle the case
437     where it is run multiple times on the same zone.  If this hook exits
438     with a non-zero exit status, the uninstall operation will fail.

440     The following replacements are performed:

442       %z        Name of zone
443       %R        Zonepath of zone
444       Additional arguments, if any, are appended.

446     It has no attributes.
447 -->
448 <!ELEMENT preuninstall  (#PCDATA) >
449 <!ATTLIST preuninstall>

451 <!--
452   uninstall
453     Identifies the hook to invoke when uninstalling a zone.  The zone will
454     be in the "incomplete" state when this hook is run.

456     If this hook exits with a non-zero exit status, the uninstall operation
457     will fail and the zone will be left in the "incomplete" state, otherwise
```



```
458     the state will be changed to "configured".

460     The following replacements are performed:

462       %z        Name of zone
463       %R        Zonepath of zone
464       Additional arguments, if any, are appended.

466     If no hook is provided, the internal zoneadm uninstall code will be used.
467 -->
468 <!ELEMENT uninstall     (#PCDATA) >
469 <!ATTLIST uninstall>

471 <!--
472   presnap
473     Identifies the hook to invoke before snapshotting a zone using the
474     built-in ZFS clone support.

476     If this hook exits with a non-zero exit status, the snapshot operation
477     will fail and the zfs clone operation will fail.

479     The following replacements are performed:

481       %z        Name of zone
482       %R        Zonepath of zone
483 -->
484 <!ELEMENT presnap       (#PCDATA) >
485 <!ATTLIST presnap>

487 <!--
488   postsnap
489     Identifies the hook to invoke after snapshotting a zone using the
490     built-in ZFS clone support.

492     If this hook exits with a non-zero exit status, the zfs clone operation
493     will fail.

495     The following replacements are performed:

497       %z        Name of zone
498       %R        Zonepath of zone
499 -->
500 <!ELEMENT postsnap      (#PCDATA) >
501 <!ATTLIST postsnap>

503 <!--
504   validatesnap
505     Identifies the hook to invoke to validate a snapshot of a zone using the
506     built-in ZFS clone support.  This will validate a snapshot that was
507     explicitly specified to the clone command when the user wants to
508     re-use a snapshot from an earlier clone operation.

510     If this hook exits with a non-zero exit status, the snapshot validation
511     operation will fail, meaning the zfs snapshot cannot be used to install
512     the zone.

514     The following replacements are performed:

516       %z        Name of zone
517       %R        Zonepath of zone
518       1st arg   snapshot name
519       2nd arg   snapshot path
520 -->
521 <!ELEMENT validatesnap  (#PCDATA) >
522 <!ATTLIST validatesnap>
```

```
<!--
  prestatechange
    Identifies the hook to invoke before zoneadmd makes a state change.
    If this hook exits with a non-zero exit status, the action failed
    and no further state change activity will take place.

    The following replacements are performed:

      %z        Name of zone
      %R        Zonepath of zone
      1st arg   integer representing current state of zone
                        2 - installed
                        3 - ready
                        4 - running
                        5 - shutting down
                        6 - down
                        7 - mounted
      2nd arg   integer representing transition command
                        0 - ready
                        1 - boot
                        4 - halt
      3rd arg   Alternate root (zonepath is mounted under this root)
                        empty string if zone not mounted under alternate root
-->
<!ELEMENT prestatechange        (#PCDATA) >
<!ATTLIST prestatechange>

<!--
  poststatechange
    Identifies the hook to invoke after zoneadmd makes a successful state
    change.  If this hook exits with a non-zero exit status, the action failed
    and zoneadmd treats the overall state change as failed, although
    all of the actions up to running the hook will have taken place.

    The following replacements are performed:

      %z        Name of zone
      %R        Zonepath of zone
    See prestatechange comment for 1st, 2nd and 3rd argument values.
-->
<!ELEMENT poststatechange       (#PCDATA) >
<!ATTLIST poststatechange>

<!--
  query
    Identifies a hook which can be called to get brand-specific information
    about the zone.  There is no specific place in zones where this is called,
    calls within the zone infrastructure can be added as needed.

    One example of the use of this hook is to query the implicit ZFS datasets
    supported by the brand.

    If this hook exits with a non-zero exit status, the query failed,
    although in general, this hook shouldn't return non-zero.

    The following replacements are performed:

      %z        Name of zone
      %R        Zonepath of zone
      1st arg   Arbitrary string which the hook can use to determine what
                data to return.  Brands implementing this hook should be
                tolerant of arguments they don't support and simply do
                nothing.
-->
<!ELEMENT query (#PCDATA) >
<!ATTLIST query>
```



```
<!--
  privilege

    Add a privilege to the default, prohibited, or required set for all
    zones of this brand with ip-type matched.  If a privilege is added
    to the default set all zones of this brand with ip-type matched on
    the system will inherit this privilege unless the privilege is
    removed via limitpriv in zonecfg(1m).  If a privilege is added to
    the prohibited set it can not be added to any zones with ip-type
    matched via limitpriv in zonecfg(1m).  If a privilege is added to
    the required set then all zones of this brand with ip-type matched
    on the system will inherit this privilege and it can't be removed via
    limitpriv in zonecfg(1m).

    Its attributes are
      set       The name of the set the privilege should go into.
      name      The name of the privilege.
      ip-type   Optional, indicates that adding of the privilege to the
                set only applies to certain IP types. Can be "shared" or
                "exclusive". If it is not specified, the default value
                "all" will be used, which means it is applicable regardless
                the IP type.

-->
<!ELEMENT privilege     (#PCDATA) >
<!ATTLIST privilege     set     ( default | prohibited | required ) #REQUIRED
                        name    CDATA #REQUIRED
                        ip-type ( shared | exclusive ) "all" >

<!--
  brand

    The toplevel container for a brand configuration.

    Its attributes are

      name      The name of the brand.  This must match the name of the
                directory in which the configuration file is stored.
-->

<!ELEMENT brand         (modname?, initname, login_cmd, forcedlogin_cmd,
                        user_cmd, install,
                        installopts?, boot?, sysboot?, halt?, shutdown?,
                        verify_cfg?, verify_adm?, postattach?, postclone?,
                        postinstall?, predetach?, attach?, detach?, clone?,
                        installopts?, boot?, sysboot?, halt?, verify_cfg?,
                        verify_adm?, postattach?, postclone?, postinstall?,
                        predetach?, attach?, detach?, clone?,
                        presnap?, postsnap?, validatesnap?,
                        preuninstall?, uninstall?,
                        prestatechange?, poststatechange?, query?,
                        privilege+)>

<!ATTLIST brand         name            CDATA #REQUIRED>
```

```
**********************************************************
   17257 Mon Feb 24 16:22:37 2014
new/usr/src/man/man1m/zoneadm.1m
2594 implement graceful shutdown for local zones in zoneadm
**********************************************************
```

```
   1 '\" te
   2 .\" Copyright 2014 Nexenta Systems, Inc. All rights reserved.
   3 .\" Copyright (c) 2009 Sun Microsystems, Inc. All Rights Reserved.
   4 .\" The contents of this file are subject to the terms of the Common Development
   5 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
   6 .\" When distributing Covered Code, include this CDDL HEADER in each file and in
   7 .TH ZONEADM 1M "Oct 30, 2013"
   6 .TH ZONEADM 1M "Feb 13, 2009"
   8 .SH NAME
   9 zoneadm \- administer zones
  10 .SH SYNOPSIS
  11 .LP
  12 .nf
  13 \fBzoneadm\fR \fB-z\fR \fIzonename\fR [\fB-u\fR \fIuuid-match\fR] \fIsubcommand\
  14      [\fIsubcommand_options\fR]
  15 .fi

  17 .LP
  18 .nf
  19 \fBzoneadm\fR [\fB-R\fR \fIroot\fR] [\fB-z\fR \fIzonename\fR] [\fB-u\fR \fIuuid-
  20      [\fIlist_options\fR]
  21 .fi

  23 .LP
  24 .nf
  25 \fBzoneadm\fR [\fB-R\fR \fIroot\fR] \fB-z\fR \fIzonename\fR [\fB-u\fR \fIuuid-ma
  26 .fi

  28 .SH DESCRIPTION
  29 .sp
  30 .LP
  31 The \fBzoneadm\fR utility is used to administer system zones. A zone is an
  32 application container that is maintained by the operating system runtime.
  33 .SH SECURITY
  34 .sp
  35 .LP
  36 Once a process has been placed in a zone other than zone \fB0\fR, the process
  37 or any of its children cannot change zones.
  38 .SH OPTIONS
  39 .sp
  40 .LP
  41 The following options are supported:
  42 .sp
  43 .ne 2
  44 .na
  45 \fB\fB-R\fR \fIroot\fR\fR
  46 .ad
  47 .sp .6
  48 .RS 4n
  49 Specify an alternate root (boot environment). This option can only be used in
  50 conjunction with the "\fBlist\fR" and "\fBmark\fR" subcommands.
  51 .RE

  53 .sp
  54 .ne 2
  55 .na
  56 \fB\fB-u\fR \fIuuid-match\fR\fR
  57 .ad
  58 .sp .6
  59 .RS 4n
  60 Unique identifier for a zone, as assigned by \fBlibuuid\fR(3LIB). If this
```



```
  61 option is present and the argument is a non-empty string, then the zone
  62 matching the \fBUUID\fR is selected instead of the one named by the \fB-z\fR
  63 option, if such a zone is present.
  64 .RE

  66 .sp
  67 .ne 2
  68 .na
  69 \fB\fB-z\fR \fIzonename\fR\fR
  70 .ad
  71 .sp .6
  72 .RS 4n
  73 String identifier for a zone.
  74 .RE

  76 .SH SUBCOMMANDS
  77 .sp
  78 .LP
  79 Subcommands which can result in destructive actions or loss of work have a
  80 \fB-F\fR flag to force the action. If input is from a terminal device, the user
  81 is prompted if such a command is given without the \fB-F\fR flag; otherwise, if
  82 such a command is given without the \fB-F\fR flag, the action is disallowed,
  83 with a diagnostic message written to standard error. If a zone installation or
  84 uninstallation is interrupted, the zone is left in the incomplete state. Use
  85 uninstall to reset such a zone back to the configured state.
  86 .sp
  87 .LP
  88 The following subcommands are supported:
  89 .sp
  90 .ne 2
  91 .na
  92 \fB\fBattach\fR [\fB-F\fR] [\fB-n\fR \fIpath\fR] [\fIbrand-specific
  93 options\fR]\fR
  94 .ad
  95 .sp .6
  96 .RS 4n
  97 The \fBattach\fR subcommand takes a zone that has been detached from one system
  98 and attaches the zone onto a new system. Therefore, it is advised (though not
  99 required) that the \fBdetach\fR subcommand should be run before the "attach"
 100 takes place. Once you have the new zone in the configured state, use the
 101 \fBattach\fR subcommand to set up the zone root instead of installing the zone
 102 as a new zone.
 103 .sp
 104 The \fB-F\fR option can be used to force the zone into the "installed" state
 105 with no validation. This option should be used with care since it can leave the
 106 zone in an unsupportable state if it was moved from a source system to a target
 107 system that is unable to properly host the zone. The \fB-n\fR option can be
 108 used to run the \fBattach\fR subcommand, without executing the command. It uses
 109 the output of the "\fBdetach\fR \fB-n\fR" subcommand as input and is useful to
 110 identify any conflicting issues, such as the network device being incompatible,
 111 and can also determine whether the host is capable of supporting the zone. The
 112 path can be "\fB-\fR", to read the input from standard input.
 113 .sp
 114 The zone's brand may include additional options that govern how the zone will
 115 be attached. See \fBbrands\fR(5) for specific brand information.
 116 .sp
 117 The zone being attached must first be configured using the \fBzonecfg\fR (see
 118 \fBzonecfg\fR(1M)) command. This does not apply when running "\fBattach\fR
 119 \fB-n\fR".
 120 .sp
 121 Use the following command to attach a zone:
 122 .sp
 123 .in +2
 124 .nf
 125 # \fBzoneadm -z my-zone attach\fR
 126 .fi
```

```
.in -2
.sp

.RE

.sp
.ne 2
.na
\fB\fBboot\fR [\fB--\fR \fIboot_options\fR]\fR
.ad
.sp .6
.RS 4n
Boot (or activate) the specified zones.
.sp
The following \fIboot_options\fR are supported:
.sp
.ne 2
.na
\fB\fB-i\fR \fIaltinit\fR\fR
.ad
.sp .6
.RS 4n
Select an alternative executable to be the primordial Process. \fIaltinit\fR is
a valid path to an executable. The default primordial process is
\fBinit\fR(1M).
.RE

.sp
.ne 2
.na
\fB\fB-m\fR \fIsmf_options\fR\fR
.ad
.sp .6
.RS 4n
The \fIsmf_options\fR include two categories of options to control booting
behavior of the service management facility: recovery options and messages
options.
.sp
Message options determine the type and amount of messages that \fBsmf\fR(5)
displays during boot. Service options determine the services which are used to
boot the system. See \fBkernel\fR(1M) for a listing of the \fB-m\fR suboptions.
.RE

.sp
.ne 2
.na
\fB\fB-s\fR\fR
.ad
.sp .6
.RS 4n
Boots only to milestone \fBsvc:/milestone/single-user:default\fR. This
milestone is equivalent to init level \fBs\fR. See \fBsvc.startd\fR(1M) and
\fBinit\fR(1M).
.RE

.RE

.sp
.ne 2
.na
\fB\fBclone\fR [\fB-m\fR \fIcopy\fR] [\fB-s\fR \fIzfs_snapshot\fR]
\fIsource_zone\fR\fR
.ad
.sp .6
.RS 4n
Install a zone by copying an existing installed zone. This subcommand is an
```



```
alternative way to install the zone.
.sp
.ne 2
.na
\fB\fB-m\fR \fIcopy\fR\fR
.ad
.sp .6
.RS 4n
Force the clone to be a copy, even if a "\fBZFS\fR clone" is possible.
.RE

.sp
.ne 2
.na
\fB\fB-s\fR \fIzfs_snapshot\fR\fR
.ad
.sp .6
.RS 4n
Specify the name of a \fBZFS\fR snapshot to use as the source of the clone. The
\fIsnapshot\fR must be a \fIsnapshot\fR of the source zone taken from a
previous "\fBzoneadm\fR clone" installation.
.RE

The source zone must be halted before this subcommand can be used.
.RE

.sp
.ne 2
.na
\fB\fBdetach\fR [\fB-n\fR]\fR
.ad
.sp .6
.RS 4n
Detach the specified zone. Detaching a zone is the first step in moving a zone
from one system to another. The full procedure to migrate a zone is that the
zone is detached, the \fIzonepath\fR directory is moved to the new host, and
then the zone is attached on the new host. Once the zone is detached, it is
left in the configured state. If you try to install or clone to a configured
zone that has been detached, you will receive an error message and the
\fBinstall\fR or \fBclone\fR subcommand will not be allowed to proceed. The
\fB-n\fR option can be used to run the \fBdetach\fR subcommand, without
executing the command. This generates the information needed for running the
"\fBattach\fR \fB-n\fR" subcommand, which is useful to identify any conflicting
issues, such as the network device being incompatible or if the host is capable
of supporting the zone. The information is sent to standard output and can be
saved to a file or piped to the "\fBattach\fR \fB-n\fR" subcommand.
.sp
Use the following command to detach a zone:
.sp
.in +2
.nf
# zoneadm -z my-zone detach
.fi
.in -2
.sp

The source zone must be halted before this subcommand can be used.
.RE

.sp
.ne 2
.na
\fB\fBhalt\fR\fR
.ad
.sp .6
.RS 4n
```

```
259 Halt the specified zones. \fBhalt\fR bypasses running the shutdown scripts
260 inside the zone. It also removes run time resources of the zone.
260 .sp
261 Use:
262 .sp
263 .in +2
264 .nf
265 zlogin \fIzone\fR shutdown
266 .fi
267 .in -2
268 .sp

270 to cleanly shutdown the zone by running the shutdown scripts.
261 .RE

263 .sp
264 .ne 2
265 .na
266 \fB\fBhelp\fR [\fIsubcommand\fR]\fR
267 .ad
268 .sp .6
269 .RS 4n
270 Display general help. If you specify \fIsubcommand\fR, displays help on
271 \fIsubcommand\fR.
272 .RE

274 .sp
275 .ne 2
276 .na
277 \fB\fBinstall\fR [\fB-x\fR \fInodataset\fR] [\fIbrand-specific options\fR]\fR
278 .ad
279 .sp .6
280 .RS 4n
281 Install the specified zone on the system. This subcommand automatically
282 attempts to verify first. It refuses to install if the verify step fails. See
283 the \fBverify\fR subcommand.
284 .sp
285 .ne 2
286 .na
287 \fB\fB-x\fR \fInodataset\fR\fR
288 .ad
289 .sp .6
290 .RS 4n
291 Do not create a \fBZFS\fR file system.
292 .RE

294 The zone's brand may include additional options that govern how the software
295 will be installed in the zone. See \fBbrands\fR(5) for specific brand
296 information.
297 .RE

299 .sp
300 .ne 2
301 .na
302 \fB\fBlist\fR [\fIlist_options\fR]\fR
303 .ad
304 .sp .6
305 .RS 4n
306 Display the name of the current zones, or the specified zone if indicated.
307 .sp
308 By default, all running zones are listed. If you use this subcommand with the
309 \fBzoneadm\fR \fB-z\fR \fIzonename\fR option, it lists only the specified zone,
310 regardless of its state. In this case, the \fB-i\fR and \fB-c\fR options are
311 disallowed.
312 .sp
313 If neither the \fB-i\fR or \fB-c\fR options are given, all running zones are
```



```
314 listed.
315 .sp
316 The following \fIlist_options\fR are supported:
317 .sp
318 .ne 2
319 .na
320 \fB\fB-c\fR\fR
321 .ad
322 .sp .6
323 .RS 4n
324 Display all configured zones. This option overides the \fB-i\fR option.
325 .RE

327 .sp
328 .ne 2
329 .na
330 \fB\fB-i\fR\fR
331 .ad
332 .sp .6
333 .RS 4n
334 Expand the display to all installed zones.
335 .RE

337 .sp
338 .ne 2
339 .na
340 \fB\fB-p\fR\fR
341 .ad
342 .sp .6
343 .RS 4n
344 Request machine parsable output. The output format is a list of lines, one per
345 zone, with colon- delimited fields. These fields are:
346 .sp
347 .in +2
348 .nf
349 zoneid:zonename:state:zonepath:uuid:brand:ip-type
350 .fi
351 .in -2
352 .sp

354 If the \fBzonepath\fR contains embedded colons, they can be escaped by a
355 backslash ("\:"), which is parsable by using the shell \fBread\fR(1) function
356 with the environmental variable \fBIFS\fR. The \fIuuid\fR value is assigned by
357 \fBlibuuid\fR(3LIB) when the zone is installed, and is useful for identifying
358 the same zone when present (or renamed) on alternate boot environments. Any
359 software that parses the output of the "\fBzoneadm list -p\fR" command must be
360 able to handle any fields that may be added in the future.
361 .sp
362 The \fB-v\fR and \fB-p\fR options are mutually exclusive. If neither \fB-v\fR
363 nor \fB-p\fR is used, just the zone name is listed.
364 .RE

366 .sp
367 .ne 2
368 .na
369 \fB\fB-v\fR\fR
370 .ad
371 .sp .6
372 .RS 4n
373 Display verbose information, including zone name, id, current state, root
374 directory, brand type, ip-type, and options.
375 .sp
376 The \fB-v\fR and \fB-p\fR options are mutually exclusive. If neither \fB-v\fR
377 nor \fB-p\fR is used, just the zone name is listed.
378 .RE
```

```
.RE

.sp
.ne 2
.na
\fB\fBmark incomplete\fR\fR
.ad
.sp .6
.RS 4n
Change the state of an installed zone to "incomplete." This command may be
useful in cases where administrative changes on the system have rendered a zone
unusable or inconsistent. This change cannot be undone (except by uninstalling
the zone).
.RE

.sp
.ne 2
.na
\fB\fBmove\fR \fInew_zonepath\fR\fR
.ad
.sp .6
.RS 4n
Move the \fIzonepath\fR to \fInew_zonepath\fR. The zone must be halted before
this subcommand can be used. The \fInew_zonepath\fR must be a local file system
and normal restrictions for \fIzonepath\fR apply.
.RE

.sp
.ne 2
.na
\fB\fBready\fR\fR
.ad
.sp .6
.RS 4n
Prepares a zone for running applications but does not start any user processes
in the zone.
.RE

.sp
.ne 2
.na
\fB\fBreboot\fR\ [\fB--\fR \fIboot_options\fR]]\fR
\fB\fBreboot\fR\fR
.ad
.sp .6
.RS 4n
Restart the zones. This is equivalent to a \fBhalt\fR \fBboot\fR sequence. This
subcommand fails if the specified zones are not active. See \fIboot\fR subcomman
for the boot options.
subcommand fails if the specified zones are not active.
.RE

.sp
.ne 2
.na
\fB\fBshutdown\fR [\fB-r\fR [\fB--\fR \fIboot_options\fR]]\fR
.ad
.sp .6
.RS 4n
Gracefully shutdown the specified zone. This subcommand waits for all zone
processes to finish; the default timeout is SCF_PROPERTY_TIMEOUT value from
the SMF service system/zones. If the \fB-r\fR option is specified, reboot the
zone. See \fIboot\fR subcommand for the boot options.
.RE

.sp
```



```
.ne 2
.na
\fB\fBuninstall [\fR\fB-F\fR\fB]\fR\fR
.ad
.sp .6
.RS 4n
Uninstall the specified zone from the system. Use this subcommand with caution.
It removes all of the files under the \fIzonepath\fR of the zone in question.
You can use the \fB-F\fR flag to force the action.
.RE

.sp
.ne 2
.na
\fB\fBverify\fR\fR
.ad
.sp .6
.RS 4n
Check to make sure the configuration of the specified zone can safely be
installed on the machine. Following is a break-down of the checks by
\fBresource/property\fR type:
.sp
.ne 2
.na
\fB\fBzonepath\fR\fR
.ad
.sp .6
.RS 4n
\fBzonepath\fR and its parent directory exist and are owned by root with
appropriate modes . The appropriate modes are that \fBzonepath\fR is \fB700\fR,
its parent is not \fBgroup\fR or \fBworld-writable\fR and so forth.
\fBzonepath\fR is not over an NFS mount. A sub-directory of the \fBzonepath\fR
named "root" does not exist.
.sp
If \fBzonepath\fR does not exist, the \fBverify\fR does not fail, but merely
warns that a subsequent install will attempt to create it with proper
permissions. A \fBverify\fR subsequent to that might fail should anything go
wrong.
.sp
\fBzonepath\fR cannot be a symbolic link.
.RE

.sp
.ne 2
.na
\fB\fBfs\fR\fR
.ad
.sp .6
.RS 4n
Any \fBfs\fR resources have their \fItype\fR value checked. An error is
reported if the value is one of \fBproc\fR, \fBmntfs\fR, \fBautofs\fR,
\fBcachefs\fR, or \fBnfs\fR or the filesystem does not have an associated mount
binary at \fB/usr/lib/fs/\fI<fstype>\fR/mount\fR.
.sp
It is an error for the \fIdirectory\fR to be a relative path.
.sp
It is an error for the path specified by \fBraw\fR to be a relative path or if
there is no \fBfsck\fR binary for a given filesystem type at
\fB/usr/lib/fs/\fI<fstype>\fR/fsck\fR. It is also an error if a corresponding
\fBfsck\fR binary exists but a \fBraw\fR path is not specified.
.RE

.sp
.ne 2
.na
\fB\fBnet\fR\fR
```

```
.ad
.sp .6
.RS 4n
All physical network interfaces exist. All network address resources are one
of:
.RS +4
.TP
.ie t \(bu
.el o
a valid IPv4 address, optionally followed by "\fB/\fR" and a prefix length;
.RE
.RS +4
.TP
.ie t \(bu
.el o
a valid IPv6 address, which must be followed by "\fB/\fR" and a prefix length;
.RE
.RS +4
.TP
.ie t \(bu
.el o
a host name which resolves to an IPv4 address.
.RE
Note that hostnames that resolve to IPv6 addresses are not supported.
.sp
The physical interface name is the network interface name.
.sp
A zone can be configured to be either exclusive-IP or shared-IP. For a
shared-IP zone, both the physical and address properties must be set. For an
exclusive-IP zone, the physical property must be set and the address property
cannot be set.
.RE

.sp
.ne 2
.na
\fB\fBrctl\fR\fR
.ad
.sp .6
.RS 4n
It also verifies that any defined resource control values are valid on the
current machine. This means that the privilege level is \fBprivileged\fR, the
limit is lower than the currently defined system value, and that the defined
action agrees with the actions that are valid for the given resource control.
.RE

.RE

.SH EXAMPLES
.LP
\fBExample 1 \fRUsing the \fB-m\fR Option
.sp
.LP
The following command illustrates the use of the \fB-m\fR option.

.sp
.in +2
.nf
# \fBzoneadm boot -- -m verbose\fR
.fi
.in -2
.sp

.LP
\fBExample 2 \fRUsing the \fB-i\fR Option
.sp
```



```
.LP
The following command illustrates the use of the \fB-i\fR option.

.sp
.in +2
.nf
# \fBzoneadm boot -- -i /sbin/init\fR
.fi
.in -2
.sp

.LP
\fBExample 3 \fRUsing the \fB-s\fR Option
.sp
.LP
The following command illustrates the use of the \fB-s\fR option.

.sp
.in +2
.nf
# \fBzoneadm boot -- -s\fR
.fi
.in -2
.sp

.SH EXIT STATUS
.sp
.LP
The following exit values are returned:
.sp
.ne 2
.na
\fB\fB0\fR\fR
.ad
.sp .6
.RS 4n
Successful completion.
.RE

.sp
.ne 2
.na
\fB\fB1\fR\fR
.ad
.sp .6
.RS 4n
An error occurred.
.RE

.sp
.ne 2
.na
\fB\fB2\fR\fR
.ad
.sp .6
.RS 4n
Invalid usage.
.RE

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

.sp
```

```
.TS
box;
c | c
l | l .
ATTRIBUTE TYPE  ATTRIBUTE VALUE
_
Interface Stability     Committed
.TE

.SH SEE ALSO
.sp
.LP
\fBread\fR(1), \fBsvcs\fR(1), \fBzlogin\fR(1), \fBzonename\fR(1),
\fBinit\fR(1M), \fBkernel\fR(1M), \fBsvcadm\fR(1M), \fBsvc.startd\fR(1M),
\fBsvc.startd\fR(1M), \fBzonecfg\fR(1M), \fBlibuuid\fR(3LIB),
\fBattributes\fR(5), \fBbrands\fR(5), \fBnative\fR(5), \fBsmf\fR(5),
\fBzones\fR(5)
.SH NOTES
.sp
.LP
The \fBzones\fR(5) service is managed by the service management facility,
\fBsmf\fR(5), under the service identifier:
.sp
.in +2
.nf
svc:/system/zones:default
.fi
.in -2
.sp

.sp
.LP
Administrative actions on this service, such as enabling, disabling, or
requesting restart, can be performed using \fBsvcadm\fR(1M). The service's
status can be queried using the \fBsvcs\fR(1) command.
.sp
.LP
The act of installing a new non-global zone is a fresh installation of the
Solaris operating system. A new installation of Solaris must not require
interaction with the user (that is, it must be "hands off"). Because of this,
packages installed in the global zone and all non-global zones cannot contain
request scripts (see \fBpkgask\fR(1M)). If a package did have a request script,
then the creation of a non-global zone could not be done without user
intervention. Any package that contains a request script is added to the global
zone only. See \fBpkgadd\fR(1M).
```

```
**********************************************************
   24029 Mon Feb 24 16:22:37 2014
new/usr/src/uts/common/sys/zone.h
2594 implement graceful shutdown for local zones in zoneadm
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */
  21 /*
  22  * Copyright (c) 2003, 2010, Oracle and/or its affiliates. All rights reserved.
  23  * Copyright 2014 Nexenta Systems, Inc. All rights reserved.
  24  */

  26 #ifndef _SYS_ZONE_H
  27 #define _SYS_ZONE_H

  29 #include <sys/types.h>
  30 #include <sys/mutex.h>
  31 #include <sys/param.h>
  32 #include <sys/rctl.h>
  33 #include <sys/ipc_rctl.h>
  34 #include <sys/pset.h>
  35 #include <sys/tsol/label.h>
  36 #include <sys/cred.h>
  37 #include <sys/netstack.h>
  38 #include <sys/uadmin.h>
  39 #include <sys/ksynch.h>
  40 #include <sys/socket_impl.h>
  41 #include <netinet/in.h>

  43 #ifdef  __cplusplus
  44 extern "C" {
  45 #endif

  47 /*
  48  * NOTE
  49  *
  50  * The contents of this file are private to the implementation of
  51  * Solaris and are subject to change at any time without notice.
  52  * Applications and drivers using these interfaces may fail to
  53  * run on future releases.
  54  */

  56 /* Available both in kernel and for user space */

  58 /* zone id restrictions and special ids */
  59 #define MAX_ZONEID      9999
  60 #define MIN_USERZONEID  1       /* lowest user-creatable zone ID */
  61 #define MIN_ZONEID      0       /* minimum zone ID on system */
```



```
  62 #define GLOBAL_ZONEID   0
  63 #define ZONEID_WIDTH    4       /* for printf */

  65 /*
  66  * Special zoneid_t token to refer to all zones.
  67  */
  68 #define ALL_ZONES       (-1)

  70 /* system call subcodes */
  71 #define ZONE_CREATE             0
  72 #define ZONE_DESTROY            1
  73 #define ZONE_GETATTR            2
  74 #define ZONE_ENTER              3
  75 #define ZONE_LIST               4
  76 #define ZONE_SHUTDOWN           5
  77 #define ZONE_LOOKUP             6
  78 #define ZONE_BOOT               7
  79 #define ZONE_VERSION            8
  80 #define ZONE_SETATTR            9
  81 #define ZONE_ADD_DATALINK       10
  82 #define ZONE_DEL_DATALINK       11
  83 #define ZONE_CHECK_DATALINK     12
  84 #define ZONE_LIST_DATALINK      13

  86 /* zone attributes */
  87 #define ZONE_ATTR_ROOT          1
  88 #define ZONE_ATTR_NAME          2
  89 #define ZONE_ATTR_STATUS        3
  90 #define ZONE_ATTR_PRIVSET       4
  91 #define ZONE_ATTR_UNIQID        5
  92 #define ZONE_ATTR_POOLID        6
  93 #define ZONE_ATTR_INITPID       7
  94 #define ZONE_ATTR_SLBL          8
  95 #define ZONE_ATTR_INITNAME      9
  96 #define ZONE_ATTR_BOOTARGS      10
  97 #define ZONE_ATTR_BRAND         11
  98 #define ZONE_ATTR_PHYS_MCAP     12
  99 #define ZONE_ATTR_SCHED_CLASS   13
 100 #define ZONE_ATTR_FLAGS         14
 101 #define ZONE_ATTR_HOSTID        15
 102 #define ZONE_ATTR_FS_ALLOWED    16
 103 #define ZONE_ATTR_NETWORK       17

 105 /* Start of the brand-specific attribute namespace */
 106 #define ZONE_ATTR_BRAND_ATTRS   32768

 108 #define ZONE_FS_ALLOWED_MAX     1024

 110 #define ZONE_EVENT_CHANNEL      "com.sun:zones:status"
 111 #define ZONE_EVENT_STATUS_CLASS "status"
 112 #define ZONE_EVENT_STATUS_SUBCLASS      "change"

 114 #define ZONE_EVENT_UNINITIALIZED        "uninitialized"
 115 #define ZONE_EVENT_INITIALIZED          "initialized"
 116 #define ZONE_EVENT_READY                "ready"
 117 #define ZONE_EVENT_RUNNING              "running"
 118 #define ZONE_EVENT_SHUTTING_DOWN        "shutting_down"

 120 #define ZONE_CB_NAME            "zonename"
 121 #define ZONE_CB_NEWSTATE        "newstate"
 122 #define ZONE_CB_OLDSTATE        "oldstate"
 123 #define ZONE_CB_TIMESTAMP       "when"
 124 #define ZONE_CB_ZONEID          "zoneid"

 126 /*
 127  * Exit values that may be returned by scripts or programs invoked by various
```

```
 128  * zone commands.
 129  *
 130  * These are defined as:
 131  *
 132  *      ZONE_SUBPROC_OK
 133  *      ===============
 134  *      The subprocess completed successfully.
 135  *
 136  *      ZONE_SUBPROC_USAGE
 137  *      ==================
 138  *      The subprocess failed with a usage message, or a usage message should
 139  *      be output in its behalf.
 140  *
 141  *      ZONE_SUBPROC_NOTCOMPLETE
 142  *      ========================
 143  *      The subprocess did not complete, but the actions performed by the
 144  *      subprocess require no recovery actions by the user.
 145  *
 146  *      For example, if the subprocess were called by "zoneadm install," the
 147  *      installation of the zone did not succeed but the user need not perform
 148  *      a "zoneadm uninstall" before attempting another install.
 149  *
 150  *      ZONE_SUBPROC_FATAL
 151  *      ==================
 152  *      The subprocess failed in a fatal manner, usually one that will require
 153  *      some type of recovery action by the user.
 154  *
 155  *      For example, if the subprocess were called by "zoneadm install," the
 156  *      installation of the zone did not succeed and the user will need to
 157  *      perform a "zoneadm uninstall" before another install attempt is
 158  *      possible.
 159  *
 160  *      The non-success exit values are large to avoid accidental collision
 161  *      with values used internally by some commands (e.g. "Z_ERR" and
 162  *      "Z_USAGE" as used by zoneadm.)
 163  */
 164 #define ZONE_SUBPROC_OK                 0
 165 #define ZONE_SUBPROC_USAGE              253
 166 #define ZONE_SUBPROC_NOTCOMPLETE        254
 167 #define ZONE_SUBPROC_FATAL              255

 169 #ifdef _SYSCALL32
 170 typedef struct {
 171         caddr32_t zone_name;
 172         caddr32_t zone_root;
 173         caddr32_t zone_privs;
 174         size32_t zone_privssz;
 175         caddr32_t rctlbuf;
 176         size32_t rctlbufsz;
 177         caddr32_t extended_error;
 178         caddr32_t zfsbuf;
 179         size32_t  zfsbufsz;
 180         int match;                      /* match level */
 181         uint32_t doi;                   /* DOI for label */
 182         caddr32_t label;                /* label associated with zone */
 183         int flags;
 184 } zone_def32;
______unchanged_portion_omitted_
 226 #define ZONE_MIN_STATE          ZONE_IS_UNINITIALIZED
 227 #define ZONE_MAX_STATE          ZONE_IS_DEAD

 229 /*
 230  * Valid commands which may be issued by zoneadm to zoneadmd.  The kernel also
 231  * communicates with zoneadmd, but only uses Z_REBOOT and Z_HALT.
 232  */
 233 typedef enum zone_cmd {
```



```
 234         Z_READY, Z_BOOT, Z_FORCEBOOT, Z_REBOOT, Z_HALT, Z_NOTE_UNINSTALLING,
 235         Z_MOUNT, Z_FORCEMOUNT, Z_UNMOUNT, Z_SHUTDOWN
 234         Z_MOUNT, Z_FORCEMOUNT, Z_UNMOUNT
 236 } zone_cmd_t;
______unchanged_portion_omitted_
```