```
**********************************************************
  191218 Fri May  2 23:39:34 2014
new/usr/src/cmd/zonecfg/zonecfg.c
1784 zone configuration passes zonecfg but not zoneadm for limitpriv property
**********************************************************
______unchanged_portion_omitted_

5708 /*
5709  * See the DTD for which attributes are required for which resources.
5710  *
5711  * This function can be called by commit_func(), which needs to save things,
5712  * in addition to the general call from parse_and_run(), which doesn't need
5713  * things saved.  Since the parameters are standardized, we distinguish by
5714  * having commit_func() call here with cmd->cmd_arg set to "save" to indicate
5715  * that a save is needed.
5716  */
5717 void
5718 verify_func(cmd_t *cmd)
5719 {
5720         struct zone_nwiftab nwiftab;
5721         struct zone_fstab fstab;
5722         struct zone_attrtab attrtab;
5723         struct zone_rctltab rctltab;
5724         struct zone_dstab dstab;
5725         struct zone_psettab psettab;
5726         struct zone_admintab admintab;
5727         char zonepath[MAXPATHLEN];
5728         char sched[MAXNAMELEN];
5729         char brand[MAXNAMELEN];
5730         char hostidp[HW_HOSTID_LEN];
5731         char fsallowedp[ZONE_FS_ALLOWED_MAX];
5732         priv_set_t *privs;
5733         char *privname = NULL;
5734         int err, ret_val = Z_OK, arg;
5735         int pset_res;
5736         boolean_t save = B_FALSE;
5737         boolean_t arg_err = B_FALSE;
5738         zone_iptype_t iptype;
5739         boolean_t has_cpu_shares = B_FALSE;
5740         boolean_t has_cpu_cap = B_FALSE;
5741         struct xif *tmp;

5743         optind = 0;
5744         while ((arg = getopt(cmd->cmd_argc, cmd->cmd_argv, "?")) != EOF) {
5745                 switch (arg) {
5746                 case '?':
5747                         longer_usage(CMD_VERIFY);
5748                         arg_err = B_TRUE;
5749                         break;
5750                 default:
5751                         short_usage(CMD_VERIFY);
5752                         arg_err = B_TRUE;
5753                         break;
5754                 }
5755         }
5756         if (arg_err)
5757                 return;

5759         if (optind > cmd->cmd_argc) {
5760                 short_usage(CMD_VERIFY);
5761                 return;
5762         }

5764         if (zone_is_read_only(CMD_VERIFY))
5765                 return;
```



```
5767         assert(cmd != NULL);

5769         if (cmd->cmd_argc > 0 && (strcmp(cmd->cmd_argv[0], "save") == 0))
5770                 save = B_TRUE;
5771         if (initialize(B_TRUE) != Z_OK)
5772                 return;

5774         if (zonecfg_get_zonepath(handle, zonepath, sizeof (zonepath)) != Z_OK &&
5775             !global_zone) {
5776                 zerr(gettext("%s not specified"), pt_to_str(PT_ZONEPATH));
5777                 ret_val = Z_REQD_RESOURCE_MISSING;
5778                 saw_error = B_TRUE;
5779         }
5780         if (strlen(zonepath) == 0 && !global_zone) {
5781                 zerr(gettext("%s cannot be empty."), pt_to_str(PT_ZONEPATH));
5782                 ret_val = Z_REQD_RESOURCE_MISSING;
5783                 saw_error = B_TRUE;
5784         }

5786         if ((err = zonecfg_get_brand(handle, brand, sizeof (brand))) != Z_OK) {
5787                 zone_perror(zone, err, B_TRUE);
5788                 return;
5789         }
5790         if ((err = brand_verify(handle)) != Z_OK) {
5791                 zone_perror(zone, err, B_TRUE);
5792                 return;
5793         }

5795         if (zonecfg_get_iptype(handle, &iptype) != Z_OK) {
5796                 zerr("%s %s", gettext("cannot get"), pt_to_str(PT_IPTYPE));
5797                 ret_val = Z_REQD_RESOURCE_MISSING;
5798                 saw_error = B_TRUE;
5799         }

5801         if ((privs = priv_allocset()) == NULL) {
5802                 zerr(gettext("%s: priv_allocset failed"), zone);
5803                 return;
5804         }
5805         if (zonecfg_get_privset(handle, privs, &privname) != Z_OK) {
5806                 zerr(gettext("%s: invalid privilege: %s"), zone, privname);
5807                 priv_freeset(privs);
5808                 free(privname);
5809                 return;
5810         }
5811         priv_freeset(privs);

5813         if (zonecfg_get_hostid(handle, hostidp,
5814             sizeof (hostidp)) == Z_INVALID_PROPERTY) {
5815                 zerr(gettext("%s: invalid hostid: %s"),
5816                     zone, hostidp);
5817                 return;
5818         }

5820         if (zonecfg_get_fs_allowed(handle, fsallowedp,
5821             sizeof (fsallowedp)) == Z_INVALID_PROPERTY) {
5822                 zerr(gettext("%s: invalid fs-allowed: %s"),
5823                     zone, fsallowedp);
5824                 return;
5825         }

5827         if ((err = zonecfg_setfsent(handle)) != Z_OK) {
5828                 zone_perror(zone, err, B_TRUE);
5829                 return;
5830         }
5831         while (zonecfg_getfsent(handle, &fstab) == Z_OK) {
5832                 check_reqd_prop(fstab.zone_fs_dir, RT_FS, PT_DIR, &ret_val);
```

```
5833                check_reqd_prop(fstab.zone_fs_special, RT_FS, PT_SPECIAL,
5834                    &ret_val);
5835                check_reqd_prop(fstab.zone_fs_type, RT_FS, PT_TYPE, &ret_val);

5837                zonecfg_free_fs_option_list(fstab.zone_fs_options);
5838        }
5839        (void) zonecfg_endfsent(handle);

5841        if ((err = zonecfg_setnwifent(handle)) != Z_OK) {
5842                zone_perror(zone, err, B_TRUE);
5843                return;
5844        }
5845        while (zonecfg_getnwifent(handle, &nwiftab) == Z_OK) {
5846                /*
5847                 * physical is required in all cases.
5848                 * A shared IP requires an address,
5849                 * and may include a default router, while
5850                 * an exclusive IP must have neither an address
5851                 * nor a default router.
5852                 * The physical interface name must be valid in all cases.
5853                 */
5854                check_reqd_prop(nwiftab.zone_nwif_physical, RT_NET,
5855                    PT_PHYSICAL, &ret_val);
5856                if (validate_net_physical_syntax(nwiftab.zone_nwif_physical) !=
5857                    Z_OK) {
5858                        saw_error = B_TRUE;
5859                        if (ret_val == Z_OK)
5860                                ret_val = Z_INVAL;
5861                }

5863                switch (iptype) {
5864                case ZS_SHARED:
5865                        check_reqd_prop(nwiftab.zone_nwif_address, RT_NET,
5866                            PT_ADDRESS, &ret_val);
5867                        if (strlen(nwiftab.zone_nwif_allowed_address) > 0) {
5868                                zerr(gettext("%s: %s cannot be specified "
5869                                    "for a shared IP type"),
5870                                    rt_to_str(RT_NET),
5871                                    pt_to_str(PT_ALLOWED_ADDRESS));
5872                                saw_error = B_TRUE;
5873                                if (ret_val == Z_OK)
5874                                        ret_val = Z_INVAL;
5875                        }
5876                        break;
5877                case ZS_EXCLUSIVE:
5878                        if (strlen(nwiftab.zone_nwif_address) > 0) {
5879                                zerr(gettext("%s: %s cannot be specified "
5880                                    "for an exclusive IP type"),
5881                                    rt_to_str(RT_NET), pt_to_str(PT_ADDRESS));
5882                                saw_error = B_TRUE;
5883                                if (ret_val == Z_OK)
5884                                        ret_val = Z_INVAL;
5885                        } else {
5886                                if (!add_nwif(&nwiftab)) {
5887                                        saw_error = B_TRUE;
5888                                        if (ret_val == Z_OK)
5889                                                ret_val = Z_INVAL;
5890                                }
5891                        }
5892                        break;
5893                }
5894        }
5895        for (tmp = xif; tmp != NULL; tmp = tmp->xif_next) {
5896                if (!tmp->xif_has_address && tmp->xif_has_defrouter) {
5897                        zerr(gettext("%s: %s for %s cannot be specified "
5898                            "without %s for an exclusive IP type"),
```



```
5899                            rt_to_str(RT_NET), pt_to_str(PT_DEFROUTER),
5900                            tmp->xif_name, pt_to_str(PT_ALLOWED_ADDRESS));
5901                        saw_error = B_TRUE;
5902                        ret_val = Z_INVAL;
5903                }
5904        }
5905        free(xif);
5906        xif = NULL;
5907        (void) zonecfg_endnwifent(handle);

5909        if ((err = zonecfg_setrctlent(handle)) != Z_OK) {
5910                zone_perror(zone, err, B_TRUE);
5911                return;
5912        }
5913        while (zonecfg_getrctlent(handle, &rctltab) == Z_OK) {
5914                check_reqd_prop(rctltab.zone_rctl_name, RT_RCTL, PT_NAME,
5915                    &ret_val);

5917                if (strcmp(rctltab.zone_rctl_name, "zone.cpu-shares") == 0)
5918                        has_cpu_shares = B_TRUE;

5920                if (strcmp(rctltab.zone_rctl_name, "zone.cpu-cap") == 0)
5921                        has_cpu_cap = B_TRUE;

5923                if (rctltab.zone_rctl_valptr == NULL) {
5924                        zerr(gettext("%s: no %s specified"),
5925                            rt_to_str(RT_RCTL), pt_to_str(PT_VALUE));
5926                        saw_error = B_TRUE;
5927                        if (ret_val == Z_OK)
5928                                ret_val = Z_REQD_PROPERTY_MISSING;
5929                } else {
5930                        zonecfg_free_rctl_value_list(rctltab.zone_rctl_valptr);
5931                }
5932        }
5933        (void) zonecfg_endrctlent(handle);

5935        if ((pset_res = zonecfg_lookup_pset(handle, &psettab)) == Z_OK &&
5936            has_cpu_shares) {
5937                zerr(gettext("%s zone.cpu-shares and %s are incompatible."),
5938                    rt_to_str(RT_RCTL), rt_to_str(RT_DCPU));
5939                saw_error = B_TRUE;
5940                if (ret_val == Z_OK)
5941                        ret_val = Z_INCOMPATIBLE;
5942        }

5944        if (has_cpu_shares && zonecfg_get_sched_class(handle, sched,
5945            sizeof (sched)) == Z_OK && strlen(sched) > 0 &&
5946            strcmp(sched, "FSS") != 0) {
5947                zerr(gettext("WARNING: %s zone.cpu-shares and %s=%s are "
5948                    "incompatible"),
5949                    rt_to_str(RT_RCTL), rt_to_str(RT_SCHED), sched);
5950                saw_error = B_TRUE;
5951                if (ret_val == Z_OK)
5952                        ret_val = Z_INCOMPATIBLE;
5953        }

5955        if (pset_res == Z_OK && has_cpu_cap) {
5956                zerr(gettext("%s zone.cpu-cap and the %s are incompatible."),
5957                    rt_to_str(RT_RCTL), rt_to_str(RT_DCPU));
5958                saw_error = B_TRUE;
5959                if (ret_val == Z_OK)
5960                        ret_val = Z_INCOMPATIBLE;
5961        }

5963        if ((err = zonecfg_setattrent(handle)) != Z_OK) {
5964                zone_perror(zone, err, B_TRUE);
```

```
5965                    return;
5966            }
5967            while (zonecfg_getattrent(handle, &attrtab) == Z_OK) {
5968                    check_reqd_prop(attrtab.zone_attr_name, RT_ATTR, PT_NAME,
5969                        &ret_val);
5970                    check_reqd_prop(attrtab.zone_attr_type, RT_ATTR, PT_TYPE,
5971                        &ret_val);
5972                    check_reqd_prop(attrtab.zone_attr_value, RT_ATTR, PT_VALUE,
5973                        &ret_val);
5974            }
5975            (void) zonecfg_endattrent(handle);

5977            if ((err = zonecfg_setdsent(handle)) != Z_OK) {
5978                    zone_perror(zone, err, B_TRUE);
5979                    return;
5980            }
5981            while (zonecfg_getdsent(handle, &dstab) == Z_OK) {
5982                    if (strlen(dstab.zone_dataset_name) == 0) {
5983                            zerr("%s: %s %s", rt_to_str(RT_DATASET),
5984                                pt_to_str(PT_NAME), gettext("not specified"));
5985                            saw_error = B_TRUE;
5986                            if (ret_val == Z_OK)
5987                                    ret_val = Z_REQD_PROPERTY_MISSING;
5988                    } else if (!zfs_name_valid(dstab.zone_dataset_name,
5989                        ZFS_TYPE_FILESYSTEM)) {
5990                            zerr("%s: %s %s", rt_to_str(RT_DATASET),
5991                                pt_to_str(PT_NAME), gettext("invalid"));
5992                            saw_error = B_TRUE;
5993                            if (ret_val == Z_OK)
5994                                    ret_val = Z_BAD_PROPERTY;
5995                    }

5997            }
5998            (void) zonecfg_enddsent(handle);

6000            if ((err = zonecfg_setadminent(handle)) != Z_OK) {
6001                    zone_perror(zone, err, B_TRUE);
6002                    return;
6003            }
6004            while (zonecfg_getadminent(handle, &admintab) == Z_OK) {
6005                    check_reqd_prop(admintab.zone_admin_user, RT_ADMIN,
6006                        PT_USER, &ret_val);
6007                    check_reqd_prop(admintab.zone_admin_auths, RT_ADMIN,
6008                        PT_AUTHS, &ret_val);
6009                    if ((ret_val == Z_OK) && (getpwnam(admintab.zone_admin_user)
6010                        == NULL)) {
6011                            zerr(gettext("%s %s is not a valid username"),
6012                                pt_to_str(PT_USER),
6013                                admintab.zone_admin_user);
6014                            ret_val = Z_BAD_PROPERTY;
6015                    }
6016                    if ((ret_val == Z_OK) && (!zonecfg_valid_auths(
6017                        admintab.zone_admin_auths, zone))) {
6018                            ret_val = Z_BAD_PROPERTY;
6019                    }
6020            }
6021            (void) zonecfg_endadminent(handle);

6023            if (!global_scope) {
6024                    zerr(gettext("resource specification incomplete"));
6025                    saw_error = B_TRUE;
6026                    if (ret_val == Z_OK)
6027                            ret_val = Z_INSUFFICIENT_SPEC;
6028            }

6030            if (save) {
```



```
6031                    if (ret_val == Z_OK) {
6032                            if ((ret_val = zonecfg_save(handle)) == Z_OK) {
6033                                    need_to_commit = B_FALSE;
6034                                    (void) strlcpy(revert_zone, zone,
6035                                        sizeof (revert_zone));
6036                            }
6037                    } else {
6038                            zerr(gettext("Zone %s failed to verify"), zone);
6039                    }
6040            }
6041            if (ret_val != Z_OK)
6042                    zone_perror(zone, ret_val, B_TRUE);
6043 }
```

*unchanged_portion_omitted_*