

```

*****
162037 Wed Oct 16 18:29:02 2013
new/usr/src/cmd/zfs/zfs_main.c
4012 Upper limit of zfs set bounds check for reservation on volumes is too lo
*****
_____unchanged_portion_omitted_____

672 /*
673 * zfs create [-p] [-o prop=value] ... fs
674 * zfs create [-ps] [-b blocksize] [-o prop=value] ... -V vol size
675 *
676 * Create a new dataset. This command can be used to create filesystems
677 * and volumes. Snapshot creation is handled by 'zfs snapshot'.
678 * For volumes, the user must specify a size to be used.
679 *
680 * The '-s' flag applies only to volumes, and indicates that we should not try
681 * to set the reservation for this volume. By default we set a reservation
682 * equal to the size for any volume. For pools with SPA_VERSION >=
683 * SPA_VERSION_REFRESERVATION, we set a reservation instead.
684 *
685 * The '-p' flag creates all the non-existing ancestors of the target first.
686 */
687 static int
688 zfs_do_create(int argc, char **argv)
689 {
690     zfs_type_t type = ZFS_TYPE_FILESYSTEM;
691     zfs_handle_t *zhp = NULL;
692     uint64_t volsize;
693     int c;
694     boolean_t noreserve = B_FALSE;
695     boolean_t bflag = B_FALSE;
696     boolean_t parents = B_FALSE;
697     int ret = 1;
698     nvlist_t *props;
699     uint64_t intval;
700     int canmount = ZFS_CANMOUNT_OFF;

702     if (nvlist_alloc(&props, NV_UNIQUE_NAME, 0) != 0)
703         nomem();

705     /* check options */
706     while ((c = getopt(argc, argv, ":V:b:so:p")) != -1) {
707         switch (c) {
708             case 'V':
709                 type = ZFS_TYPE_VOLUME;
710                 if (zfs_nicestrtonum(g_zfs, optarg, &intval) != 0) {
711                     (void) fprintf(stderr, gettext("bad volume "
712                         "size '%s': %s\n"), optarg,
713                         libzfs_error_description(g_zfs));
714                     goto error;
715                 }
717                 if (nvlist_add_uint64(props,
718                     zfs_prop_to_name(ZFS_PROP_VOLSIZE), intval) != 0)
719                     nomem();
720                 volsize = intval;
721                 break;
722             case 'p':
723                 parents = B_TRUE;
724                 break;
725             case 'b':
726                 bflag = B_TRUE;
727                 if (zfs_nicestrtonum(g_zfs, optarg, &intval) != 0) {
728                     (void) fprintf(stderr, gettext("bad volume "
729                         "block size '%s': %s\n"), optarg,
730                         libzfs_error_description(g_zfs));

```

```

731         goto error;
732     }

734     if (nvlist_add_uint64(props,
735         zfs_prop_to_name(ZFS_PROP_VOLBLOCKSIZE),
736         intval) != 0)
737         nomem();
738     break;
739 case 'o':
740     if (parseprop(props))
741         goto error;
742     break;
743 case 's':
744     noreserve = B_TRUE;
745     break;
746 case ':':
747     (void) fprintf(stderr, gettext("missing size "
748         "argument\n"));
749     goto badusage;
750 case '?':
751     (void) fprintf(stderr, gettext("invalid option '%c'\n"),
752         optopt);
753     goto badusage;
754 }
755 }

757 if ((bflag || noreserve) && type != ZFS_TYPE_VOLUME) {
758     (void) fprintf(stderr, gettext("'s' and '-b' can only be "
759         "used when creating a volume\n"));
760     goto badusage;
761 }

763 argc -= optind;
764 argv += optind;

766 /* check number of arguments */
767 if (argc == 0) {
768     (void) fprintf(stderr, gettext("missing %s argument\n"),
769         zfs_type_to_name(type));
770     goto badusage;
771 }
772 if (argc > 1) {
773     (void) fprintf(stderr, gettext("too many arguments\n"));
774     goto badusage;
775 }

777 if (type == ZFS_TYPE_VOLUME && !noreserve) {
778     zpool_handle_t *zpool_handle;
779     nvlist_t *real_props;
780     uint64_t spa_version;
781     char *p;
782     zfs_prop_t resv_prop;
783     char *strval;
784     char msg[1024];
785     uint64_t volblocksize;
786     int ncopies;

788     if (p = strchr(argv[0], '/'))
789         *p = '\0';
790     zpool_handle = zpool_open(g_zfs, argv[0]);
791     if (p != NULL)
792         *p = '/';
793     if (zpool_handle == NULL)
794         goto error;
795     spa_version = zpool_get_prop_int(zpool_handle,
796         ZPOOL_PROP_VERSION, NULL);

```

```

797     zpool_close(zpool_handle);
798     if (spa_version >= SPA_VERSION_REFRESERVATION)
799         resv_prop = ZFS_PROP_REFRESERVATION;
800     else
801         resv_prop = ZFS_PROP_RESERVATION;
802
803     (void) snprintf(msg, sizeof (msg),
804         gettext("cannot create '%s'"), argv[0]);
805     if (props && (real_props = zfs_valid_proplist(g_zfs, type,
806         props, 0, NULL, msg)) == NULL)
807         goto error;
808
809     if (nvlist_lookup_string(real_props,
810         zfs_prop_to_name(ZFS_PROP_COPIES), &strval) == 0)
811         ncopies = atoi(strval);
812     else
813         ncopies = 1;
814     if (nvlist_lookup_uint64(real_props,
815         zfs_prop_to_name(ZFS_PROP_VOLBLOCKSIZE),
816         &volblocksize) != 0)
817         volblocksize = ZVOL_DEFAULT_BLOCKSIZE;
818
819     volsize = zvol_volsize_to_reservation(volsize,
820         volblocksize, ncopies);
821     volsize = zvol_volsize_to_reservation(volsize, real_props);
822     nvlist_free(real_props);
823
824     if (nvlist_lookup_string(props, zfs_prop_to_name(resv_prop),
825         &strval) != 0) {
826         if (nvlist_add_uint64(props,
827             zfs_prop_to_name(resv_prop), volsize) != 0) {
828             nvlist_free(props);
829             nomem();
830         }
831     }
832
833     if (parents && zfs_name_valid(argv[0], type)) {
834         /*
835          * Now create the ancestors of target dataset. If the target
836          * already exists and '-p' option was used we should not
837          * complain.
838          */
839         if (zfs_dataset_exists(g_zfs, argv[0], type)) {
840             ret = 0;
841             goto error;
842         }
843         if (zfs_create_ancestors(g_zfs, argv[0]) != 0)
844             goto error;
845     }
846
847     /* pass to libzfs */
848     if (zfs_create(g_zfs, argv[0], type, props) != 0)
849         goto error;
850
851     if ((zhp = zfs_open(g_zfs, argv[0], ZFS_TYPE_DATASET)) == NULL)
852         goto error;
853
854     ret = 0;
855     /*
856      * if the user doesn't want the dataset automatically mounted,
857      * then skip the mount/share step
858      */
859     if (zfs_prop_valid_for_type(ZFS_PROP_CANMOUNT, type))
860         canmount = zfs_prop_get_int(zhp, ZFS_PROP_CANMOUNT);

```

```

862     /*
863      * Mount and/or share the new filesystem as appropriate. We provide a
864      * verbose error message to let the user know that their filesystem was
865      * in fact created, even if we failed to mount or share it.
866      */
867     if (canmount == ZFS_CANMOUNT_ON) {
868         if (zfs_mount(zhp, NULL, 0) != 0) {
869             (void) fprintf(stderr, gettext("filesystem "
870                 "successfully created, but not mounted\n"));
871             ret = 1;
872         } else if (zfs_share(zhp) != 0) {
873             (void) fprintf(stderr, gettext("filesystem "
874                 "successfully created, but not shared\n"));
875             ret = 1;
876         }
877     }
878
879 error:
880     if (zhp)
881         zfs_close(zhp);
882     nvlist_free(props);
883     return (ret);
884 badusage:
885     nvlist_free(props);
886     usage(B_FALSE);
887     return (2);
888 }

```

unchanged portion omitted

```

*****
27058 Wed Oct 16 18:29:02 2013
new/usr/src/lib/libzfs/common/libzfs.h
4012 Upper limit of zfs set bounds check for reservation on volumes is too lo
*****
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) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
24  * Copyright 2011 Nexenta Systems, Inc. All rights reserved.
25  * Copyright (c) 2012 by Delphix. All rights reserved.
26  * Copyright (c) 2012, Joyent, Inc. All rights reserved.
27  * Copyright (c) 2013 Steven Hartland. All rights reserved.
28  * Copyright 2013 DEY Storage Systems, Inc.
29 */

31 #ifndef _LIBZFS_H
32 #define _LIBZFS_H

34 #include <assert.h>
35 #include <libnvpair.h>
36 #include <sys/mnttab.h>
37 #include <sys/param.h>
38 #include <sys/types.h>
39 #include <sys/varargs.h>
40 #include <sys/fs/zfs.h>
41 #include <sys/avl.h>
42 #include <ucred.h>

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

48 /*
49  * Miscellaneous ZFS constants
50  */
51 #define ZFS_MAXNAMELEN      MAXNAMELEN
52 #define ZPOOL_MAXNAMELEN   MAXNAMELEN
53 #define ZFS_MAXPROPLEN     MAXPATHLEN
54 #define ZPOOL_MAXPROPLEN   MAXPATHLEN

56 /*
57  * libzfs errors
58  */
59 typedef enum zfs_error {
60     EZFS_SUCCESS = 0,      /* no error -- success */
61     EZFS_NOMEM = 2000,    /* out of memory */

```

```

62     EZFS_BADPROP,         /* invalid property value */
63     EZFS_PROPREADONLY,   /* cannot set readonly property */
64     EZFS_PROPTYPE,       /* property does not apply to dataset type */
65     EZFS_PROPNONINHERIT, /* property is not inheritable */
66     EZFS_PROPSPACE,      /* bad quota or reservation */
67     EZFS_BADTYPE,        /* dataset is not of appropriate type */
68     EZFS_BUSY,           /* pool or dataset is busy */
69     EZFS_EXISTS,         /* pool or dataset already exists */
70     EZFS_NOENT,          /* no such pool or dataset */
71     EZFS_BADSTREAM,      /* bad backup stream */
72     EZFS_DSREADONLY,     /* dataset is readonly */
73     EZFS_VOLTOOBIG,      /* volume is too large for 32-bit system */
74     EZFS_INVALIDNAME,    /* invalid dataset name */
75     EZFS_BADRESTORE,     /* unable to restore to destination */
76     EZFS_BADBACKUP,     /* backup failed */
77     EZFS_BADTARGET,     /* bad attach/detach/replace target */
78     EZFS_NODEVICE,      /* no such device in pool */
79     EZFS_BADDEV,        /* invalid device to add */
80     EZFS_NOREPLICAS,    /* no valid replicas */
81     EZFS_RESILVERING,    /* currently resilvering */
82     EZFS_BADVERSION,     /* unsupported version */
83     EZFS_POOLUNAVAIL,   /* pool is currently unavailable */
84     EZFS_DEVOVERFLOW,   /* too many devices in one vdev */
85     EZFS_BADPATH,       /* must be an absolute path */
86     EZFS_CROSTARGET,    /* rename or clone across pool or dataset */
87     EZFS_ZONED,         /* used improperly in local zone */
88     EZFS_MOUNTFAILED,   /* failed to mount dataset */
89     EZFS_UMOUNTFAILED,  /* failed to unmount dataset */
90     EZFS_UNSHARENFSAILED, /* unshare(1M) failed */
91     EZFS_SHARENFSAILED, /* share(1M) failed */
92     EZFS_PERM,          /* permission denied */
93     EZFS_NOSPC,         /* out of space */
94     EZFS_FAULT,         /* bad address */
95     EZFS_IO,            /* I/O error */
96     EZFS_INTR,          /* signal received */
97     EZFS_ISSPARE,       /* device is a hot spare */
98     EZFS_INVALIDCONFIG, /* invalid vdev configuration */
99     EZFS_RECURSIVE,     /* recursive dependency */
100    EZFS_NOHISTORY,      /* no history object */
101    EZFS_POOLPROPS,      /* couldn't retrieve pool props */
102    EZFS_POOL_NOTSUP,    /* ops not supported for this type of pool */
103    EZFS_POOL_INVALIDARG, /* invalid argument for this pool operation */
104    EZFS_NAMETOOLONG,   /* dataset name is too long */
105    EZFS_OPENFAILED,    /* open of device failed */
106    EZFS_NOCAP,         /* couldn't get capacity */
107    EZFS_LABELFAILED,   /* write of label failed */
108    EZFS_BADWHO,        /* invalid permission who */
109    EZFS_BADPERM,       /* invalid permission */
110    EZFS_BADPERMSET,    /* invalid permission set name */
111    EZFS_NODELEGATION,  /* delegated administration is disabled */
112    EZFS_UNSHARESMBFAILED, /* failed to unshare over smb */
113    EZFS_SHARESMBFAILED, /* failed to share over smb */
114    EZFS_BADCACHE,      /* bad cache file */
115    EZFS_ISL2CACHE,     /* device is for the level 2 ARC */
116    EZFS_VDEVNOTSUP,    /* unsupported vdev type */
117    EZFS_NOTSUP,        /* ops not supported on this dataset */
118    EZFS_ACTIVE_SPARE,  /* pool has active shared spare devices */
119    EZFS_UNPLAYED_LOGS, /* log device has unplayed logs */
120    EZFS_REFTAG_RELE,   /* snapshot release: tag not found */
121    EZFS_REFTAG_HOLD,  /* snapshot hold: tag already exists */
122    EZFS_TAGTOOLONG,   /* snapshot hold/rele: tag too long */
123    EZFS_PIPEFAILED,   /* pipe create failed */
124    EZFS_THREADCREATEFAILED, /* thread create failed */
125    EZFS_POSTSPLIT_ONLINE, /* onlining a disk after splitting it */
126    EZFS_SCRUBBING,    /* currently scrubbing */
127    EZFS_NO_SCRUB,     /* no active scrub */

```

```
128     EZFS_DIFF,           /* general failure of zfs diff */
129     EZFS_DIFFDATA,       /* bad zfs diff data */
130     EZFS_POOLREADONLY,   /* pool is in read-only mode */
131     EZFS_UNKNOWN
132 } zfs_error_t;
unchanged_portion_omitted

592 typedef boolean_t (snapfilter_cb_t)(zfs_handle_t *, void *);

594 extern int zfs_send(zfs_handle_t *, const char *, const char *,
595     sendflags_t *, int, snapfilter_cb_t, void *, nvlist_t **);

597 extern int zfs_promote(zfs_handle_t *);
598 extern int zfs_hold(zfs_handle_t *, const char *, const char *,
599     boolean_t, int);
600 extern int zfs_hold_nvl(zfs_handle_t *, int, nvlist_t *);
601 extern int zfs_release(zfs_handle_t *, const char *, const char *, boolean_t);
602 extern int zfs_get_holds(zfs_handle_t *, nvlist_t **);
603 extern uint64_t zvol_volsize_to_reservation(uint64_t volsize,
604     uint64_t volblocksize, int ncopies);
602 extern uint64_t zvol_volsize_to_reservation(uint64_t, nvlist_t *);

606 typedef int (*zfs_userspace_cb_t)(void *arg, const char *domain,
607     uid_t rid, uint64_t space);

609 extern int zfs_userspace(zfs_handle_t *, zfs_userquota_prop_t,
610     zfs_userspace_cb_t, void *);

612 extern int zfs_get_fsacl(zfs_handle_t *, nvlist_t **);
613 extern int zfs_set_fsacl(zfs_handle_t *, boolean_t, nvlist_t *);

615 typedef struct recvflags {
616     /* print informational messages (ie, -v was specified) */
617     boolean_t verbose;

619     /* the destination is a prefix, not the exact fs (ie, -d) */
620     boolean_t isprefix;

622     /*
623      * Only the tail of the sent snapshot path is appended to the
624      * destination to determine the received snapshot name (ie, -e).
625      */
626     boolean_t istail;

628     /* do not actually do the recv, just check if it would work (ie, -n) */
629     boolean_t dryrun;

631     /* rollback/destroy filesystems as necessary (eg, -F) */
632     boolean_t force;

634     /* set "canmount=off" on all modified filesystems */
635     boolean_t canmountoff;

637     /* byteswap flag is used internally; callers need not specify */
638     boolean_t byteswap;

640     /* do not mount file systems as they are extracted (private) */
641     boolean_t nomount;
642 } recvflags_t;
unchanged_portion_omitted
```

```

*****
111439 Wed Oct 16 18:29:02 2013
new/usr/src/lib/libzfs/common/libzfs_dataset.c
4012 Upper limit of zfs set bounds check for reservation on volumes is too lo
*****
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) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
24  * Copyright (c) 2013 by Delphix. All rights reserved.
25  * Copyright 2013 DEY Storage Systems, Inc.
26  * Copyright (c) 2012 DEY Storage Systems, Inc. All rights reserved.
27  * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
28  * Copyright (c) 2013 Martin Matuska. All rights reserved.
29  * Copyright (c) 2013 Steven Hartland. All rights reserved.
30 */

31 #include <ctype.h>
32 #include <errno.h>
33 #include <libintl.h>
34 #include <math.h>
35 #include <stdio.h>
36 #include <stdlib.h>
37 #include <strings.h>
38 #include <unistd.h>
39 #include <stddef.h>
40 #include <zone.h>
41 #include <fcntl.h>
42 #include <sys/mntent.h>
43 #include <sys/mount.h>
44 #include <priv.h>
45 #include <pwd.h>
46 #include <grp.h>
47 #include <stddef.h>
48 #include <ucred.h>
49 #include <idmap.h>
50 #include <aclutils.h>
51 #include <directory.h>

53 #include <sys/dnode.h>
54 #include <sys/spa.h>
55 #include <sys/zap.h>
56 #include <libzfs.h>

58 #include "zfs_namecheck.h"
59 #include "zfs_prop.h"
60 #include "libzfs_impl.h"

```

```

61 #include "zfs_deleg.h"

63 static int userquota_propname_decode(const char *propname, boolean_t zoned,
64   zfs_userquota_prop_t *typep, char *domain, int domainlen, uint64_t *ridp);

66 /*
67  * Given a single type (not a mask of types), return the type in a human
68  * readable form.
69  */
70 const char *
71 zfs_type_to_name(zfs_type_t type)
72 {
73     switch (type) {
74     case ZFS_TYPE_FILESYSTEM:
75         return (dgettext(TEXT_DOMAIN, "filesystem"));
76     case ZFS_TYPE_SNAPSHOT:
77         return (dgettext(TEXT_DOMAIN, "snapshot"));
78     case ZFS_TYPE_VOLUME:
79         return (dgettext(TEXT_DOMAIN, "volume"));
80     }

82     return (NULL);
83 }

    unchanged portion omitted

789 /*
790  * Given an nvlist of properties to set, validates that they are correct, and
791  * parses any numeric properties (index, boolean, etc) if they are specified as
792  * strings.
793  */
794 nvlist_t *
795 zfs_valid_proplist(libzfs_handle_t *hdl, zfs_type_t type, nvlist_t *nvl,
796   uint64_t zoned, zfs_handle_t *zhp, const char *errbuf)
797 {
798     nvpair_t *elem;
799     uint64_t intval;
800     char *strval;
801     zfs_prop_t prop;
802     nvlist_t *ret;
803     int chosen_normal = -1;
804     int chosen_utf = -1;

806     if (nvlist_alloc(&ret, NV_UNIQUE_NAME, 0) != 0) {
807         (void) no_memory(hdl);
808         return (NULL);
809     }

811     /*
812      * Make sure this property is valid and applies to this type.
813      */

815     elem = NULL;
816     while ((elem = nvlist_next_nvpair(nvl, elem)) != NULL) {
817         const char *propname = nvpair_name(elem);

819         prop = zfs_name_to_prop(propname);
820         if (prop == ZPROP_INVALID && zfs_prop_user(propname)) {
821             /*
822              * This is a user property: make sure it's a
823              * string, and that it's less than ZAP_MAXNAMELEN.
824              */
825             if (nvpair_type(elem) != DATA_TYPE_STRING) {
826                 zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
827                   "'%s' must be a string"), propname);
828                 (void) zfs_error(hdl, EZFS_BADPROP, errbuf);
829                 goto error;

```

```

830     }
831
832     if (strlen(nvpair_name(elem)) >= ZAP_MAXNAMELEN) {
833         zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
834             "property name '%s' is too long"),
835             propname);
836         (void) zfs_error(hdl, EZFS_BADPROP, errbuf);
837         goto error;
838     }
839
840     (void) nvpair_value_string(elem, &strval);
841     if (nvlist_add_string(ret, propname, strval) != 0) {
842         (void) no_memory(hdl);
843         goto error;
844     }
845     continue;
846 }
847
848 /*
849  * Currently, only user properties can be modified on
850  * snapshots.
851  */
852 if (type == ZFS_TYPE_SNAPSHOT) {
853     zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
854         "this property can not be modified for snapshots"));
855     (void) zfs_error(hdl, EZFS_PROPTYPE, errbuf);
856     goto error;
857 }
858
859 if (prop == ZPROP_INVALID && zfs_prop_userquota(propname)) {
860     zfs_userquota_prop_t uqtype;
861     char newpropname[128];
862     char domain[128];
863     uint64_t rid;
864     uint64_t valary[3];
865
866     if (userquota_propname_decode(propname, zoned,
867         &uqtype, domain, sizeof(domain), &rid) != 0) {
868         zfs_error_aux(hdl,
869             dgettext(TEXT_DOMAIN,
870                 "'%s' has an invalid user/group name"),
871                 propname);
872         (void) zfs_error(hdl, EZFS_BADPROP, errbuf);
873         goto error;
874     }
875
876     if (uqtype != ZFS_PROP_USERQUOTA &&
877         uqtype != ZFS_PROP_GROUPQUOTA) {
878         zfs_error_aux(hdl,
879             dgettext(TEXT_DOMAIN, "'%s' is readonly"),
880             propname);
881         (void) zfs_error(hdl, EZFS_PROPREADONLY,
882             errbuf);
883         goto error;
884     }
885
886     if (nvpair_type(elem) == DATA_TYPE_STRING) {
887         (void) nvpair_value_string(elem, &strval);
888         if (strcmp(strval, "none") == 0) {
889             intval = 0;
890         } else if (zfs_nicestrtonum(hdl,
891             strval, &intval) != 0) {
892             (void) zfs_error(hdl,
893                 EZFS_BADPROP, errbuf);
894             goto error;
895         }

```

```

896     } else if (nvpair_type(elem) ==
897         DATA_TYPE_UINT64) {
898         (void) nvpair_value_uint64(elem, &intval);
899         if (intval == 0) {
900             zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
901                 "use 'none' to disable "
902                 "userquota/groupquota"));
903             goto error;
904         }
905     } else {
906         zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
907             "'%s' must be a number"), propname);
908         (void) zfs_error(hdl, EZFS_BADPROP, errbuf);
909         goto error;
910     }
911
912     /*
913     * Encode the prop name as
914     * userquota@<hex-rid>-domain, to make it easy
915     * for the kernel to decode.
916     */
917     (void) snprintf(newpropname, sizeof(newpropname),
918         "%s%llx-%s", zfs_userquota_prop_prefixes[uqtype],
919         (longlong_t)rid, domain);
920     valary[0] = uqtype;
921     valary[1] = rid;
922     valary[2] = intval;
923     if (nvlist_add_uint64_array(ret, newpropname,
924         valary, 3) != 0) {
925         (void) no_memory(hdl);
926         goto error;
927     }
928     continue;
929 } else if (prop == ZPROP_INVALID && zfs_prop_written(propname)) {
930     zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
931         "'%s' is readonly"),
932         propname);
933     (void) zfs_error(hdl, EZFS_PROPREADONLY, errbuf);
934     goto error;
935 }
936
937 if (prop == ZPROP_INVALID) {
938     zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
939         "invalid property '%s'"), propname);
940     (void) zfs_error(hdl, EZFS_BADPROP, errbuf);
941     goto error;
942 }
943
944 if (!zfs_prop_valid_for_type(prop, type)) {
945     zfs_error_aux(hdl,
946         dgettext(TEXT_DOMAIN, "'%s' does not "
947             "apply to datasets of this type"), propname);
948     (void) zfs_error(hdl, EZFS_PROPTYPE, errbuf);
949     goto error;
950 }
951
952 if (zfs_prop_readonly(prop) &&
953     (!zfs_prop_setonce(prop) || zhp != NULL)) {
954     zfs_error_aux(hdl,
955         dgettext(TEXT_DOMAIN, "'%s' is readonly"),
956         propname);
957     (void) zfs_error(hdl, EZFS_PROPREADONLY, errbuf);
958     goto error;
959 }
960
961 if (zprop_parse_value(hdl, elem, prop, type, ret,

```

```

962         &strval, &intval, errbuf) != 0)
963         goto error;

965     /*
966     * Perform some additional checks for specific properties.
967     */
968     switch (prop) {
969     case ZFS_PROP_VERSION:
970     {
971         int version;

973         if (zhp == NULL)
974             break;
975         version = zfs_prop_get_int(zhp, ZFS_PROP_VERSION);
976         if (intval < version) {
977             zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
978                 "Can not downgrade; already at version %u"),
979                 version);
980             (void) zfs_error(hdl, EZFS_BADPROP, errbuf);
981             goto error;
982         }
983         break;
984     }

986     case ZFS_PROP_RECORDSIZE:
987     case ZFS_PROP_VOLBLOCKSIZE:
988         /* must be power of two within SPA_{MIN,MAX}BLOCKSIZE */
989         if (intval < SPA_MINBLOCKSIZE ||
990             intval > SPA_MAXBLOCKSIZE || !ISP2(intval)) {
991             zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
992                 "'%s' must be power of 2 from %u "
993                 "to %uk"), propname,
994                 (uint_t)SPA_MINBLOCKSIZE,
995                 (uint_t)SPA_MAXBLOCKSIZE >> 10);
996             (void) zfs_error(hdl, EZFS_BADPROP, errbuf);
997             goto error;
998         }
999         break;

1001     case ZFS_PROP_MLSLABEL:
1002     {
1003         /*
1004         * Verify the mlslabel string and convert to
1005         * internal hex label string.
1006         */

1008         m_label_t *new_sl;
1009         char *hex = NULL; /* internal label string */

1011         /* Default value is already OK. */
1012         if (strcasecmp(strval, ZFS_MLSLABEL_DEFAULT) == 0)
1013             break;

1015         /* Verify the label can be converted to binary form */
1016         if (((new_sl = m_label_alloc(MAC_LABEL)) == NULL) ||
1017             (str_to_label(strval, &new_sl, MAC_LABEL,
1018                 L_NO_CORRECTION, NULL) == -1)) {
1019             goto badlabel;
1020         }

1022         /* Now translate to hex internal label string */
1023         if (label_to_str(new_sl, &hex, M_INTERNAL,
1024             DEF_NAMES) != 0) {
1025             if (hex)
1026                 free(hex);
1027             goto badlabel;

```

```

1028     }
1029     m_label_free(new_sl);

1031     /* If string is already in internal form, we're done. */
1032     if (strcmp(strval, hex) == 0) {
1033         free(hex);
1034         break;
1035     }

1037     /* Replace the label string with the internal form. */
1038     (void) nvlist_remove(ret, zfs_prop_to_name(prop),
1039         DATA_TYPE_STRING);
1040     verify(nvlist_add_string(ret, zfs_prop_to_name(prop),
1041         hex) == 0);
1042     free(hex);

1044     break;

1046 badlabel:
1047     zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
1048         "invalid mlslabel '%s'", strval);
1049     (void) zfs_error(hdl, EZFS_BADPROP, errbuf);
1050     m_label_free(new_sl); /* OK if null */
1051     goto error;

1053     }

1055     case ZFS_PROP_MOUNTPOINT:
1056     {
1057         namecheck_err_t why;

1059         if (strcmp(strval, ZFS_MOUNTPOINT_NONE) == 0 ||
1060             strcmp(strval, ZFS_MOUNTPOINT_LEGACY) == 0)
1061             break;

1063         if (mountpoint_namecheck(strval, &why)) {
1064             switch (why) {
1065             case NAME_ERR_LEADING_SLASH:
1066                 zfs_error_aux(hdl,
1067                     dgettext(TEXT_DOMAIN,
1068                         "'%s' must be an absolute path, "
1069                         "'none', or 'legacy'"), propname);
1070                 break;
1071             case NAME_ERR_TOOLONG:
1072                 zfs_error_aux(hdl,
1073                     dgettext(TEXT_DOMAIN,
1074                         "component of '%s' is too long"),
1075                     propname);
1076                 break;
1077             }
1078             (void) zfs_error(hdl, EZFS_BADPROP, errbuf);
1079             goto error;
1080         }
1081     }

1083     /*FALLTHRU*/

1085     case ZFS_PROP_SHARESMB:
1086     case ZFS_PROP_SHARENFVS:
1087         /*
1088         * For the mountpoint and sharenfs or sharesmb
1089         * properties, check if it can be set in a
1090         * global/non-global zone based on
1091         * the zoned property value:
1092         *
1093         *
1094         *
1095         *
1096         *
1097         *
1098         *
1099         *
1100         *
1101         *
1102         *
1103         *
1104         *
1105         *
1106         *
1107         *
1108         *
1109         *
1110         *
1111         *
1112         *
1113         *
1114         *
1115         *
1116         *
1117         *
1118         *
1119         *
1120         *
1121         *
1122         *
1123         *
1124         *
1125         *
1126         *
1127         *
1128         *
1129         *
1130         *
1131         *
1132         *
1133         *
1134         *
1135         *
1136         *
1137         *
1138         *
1139         *
1140         *
1141         *
1142         *
1143         *
1144         *
1145         *
1146         *
1147         *
1148         *
1149         *
1150         *
1151         *
1152         *
1153         *
1154         *
1155         *
1156         *
1157         *
1158         *
1159         *
1160         *
1161         *
1162         *
1163         *
1164         *
1165         *
1166         *
1167         *
1168         *
1169         *
1170         *
1171         *
1172         *
1173         *
1174         *
1175         *
1176         *
1177         *
1178         *
1179         *
1180         *
1181         *
1182         *
1183         *
1184         *
1185         *
1186         *
1187         *
1188         *
1189         *
1190         *
1191         *
1192         *
1193         *
1194         *
1195         *
1196         *
1197         *
1198         *
1199         *
1200         *
1201         *
1202         *
1203         *
1204         *
1205         *
1206         *
1207         *
1208         *
1209         *
1210         *
1211         *
1212         *
1213         *
1214         *
1215         *
1216         *
1217         *
1218         *
1219         *
1220         *
1221         *
1222         *
1223         *
1224         *
1225         *
1226         *
1227         *
1228         *
1229         *
1230         *
1231         *
1232         *
1233         *
1234         *
1235         *
1236         *
1237         *
1238         *
1239         *
1240         *
1241         *
1242         *
1243         *
1244         *
1245         *
1246         *
1247         *
1248         *
1249         *
1250         *
1251         *
1252         *
1253         *
1254         *
1255         *
1256         *
1257         *
1258         *
1259         *
1260         *
1261         *
1262         *
1263         *
1264         *
1265         *
1266         *
1267         *
1268         *
1269         *
1270         *
1271         *
1272         *
1273         *
1274         *
1275         *
1276         *
1277         *
1278         *
1279         *
1280         *
1281         *
1282         *
1283         *
1284         *
1285         *
1286         *
1287         *
1288         *
1289         *
1290         *
1291         *
1292         *
1293         *
1294         *
1295         *
1296         *
1297         *
1298         *
1299         *
1300         *
1301         *
1302         *
1303         *
1304         *
1305         *
1306         *
1307         *
1308         *
1309         *
1310         *
1311         *
1312         *
1313         *
1314         *
1315         *
1316         *
1317         *
1318         *
1319         *
1320         *
1321         *
1322         *
1323         *
1324         *
1325         *
1326         *
1327         *
1328         *
1329         *
1330         *
1331         *
1332         *
1333         *
1334         *
1335         *
1336         *
1337         *
1338         *
1339         *
1340         *
1341         *
1342         *
1343         *
1344         *
1345         *
1346         *
1347         *
1348         *
1349         *
1350         *
1351         *
1352         *
1353         *
1354         *
1355         *
1356         *
1357         *
1358         *
1359         *
1360         *
1361         *
1362         *
1363         *
1364         *
1365         *
1366         *
1367         *
1368         *
1369         *
1370         *
1371         *
1372         *
1373         *
1374         *
1375         *
1376         *
1377         *
1378         *
1379         *
1380         *
1381         *
1382         *
1383         *
1384         *
1385         *
1386         *
1387         *
1388         *
1389         *
1390         *
1391         *
1392         *
1393         *
1394         *
1395         *
1396         *
1397         *
1398         *
1399         *
1400         *
1401         *
1402         *
1403         *
1404         *
1405         *
1406         *
1407         *
1408         *
1409         *
1410         *
1411         *
1412         *
1413         *
1414         *
1415         *
1416         *
1417         *
1418         *
1419         *
1420         *
1421         *
1422         *
1423         *
1424         *
1425         *
1426         *
1427         *
1428         *
1429         *
1430         *
1431         *
1432         *
1433         *
1434         *
1435         *
1436         *
1437         *
1438         *
1439         *
1440         *
1441         *
1442         *
1443         *
1444         *
1445         *
1446         *
1447         *
1448         *
1449         *
1450         *
1451         *
1452         *
1453         *
1454         *
1455         *
1456         *
1457         *
1458         *
1459         *
1460         *
1461         *
1462         *
1463         *
1464         *
1465         *
1466         *
1467         *
1468         *
1469         *
1470         *
1471         *
1472         *
1473         *
1474         *
1475         *
1476         *
1477         *
1478         *
1479         *
1480         *
1481         *
1482         *
1483         *
1484         *
1485         *
1486         *
1487         *
1488         *
1489         *
1490         *
1491         *
1492         *
1493         *
1494         *
1495         *
1496         *
1497         *
1498         *
1499         *
1500         *
1501         *
1502         *
1503         *
1504         *
1505         *
1506         *
1507         *
1508         *
1509         *
1510         *
1511         *
1512         *
1513         *
1514         *
1515         *
1516         *
1517         *
1518         *
1519         *
1520         *
1521         *
1522         *
1523         *
1524         *
1525         *
1526         *
1527         *
1528         *
1529         *
1530         *
1531         *
1532         *
1533         *
1534         *
1535         *
1536         *
1537         *
1538         *
1539         *
1540         *
1541         *
1542         *
1543         *
1544         *
1545         *
1546         *
1547         *
1548         *
1549         *
1550         *
1551         *
1552         *
1553         *
1554         *
1555         *
1556         *
1557         *
1558         *
1559         *
1560         *
1561         *
1562         *
1563         *
1564         *
1565         *
1566         *
1567         *
1568         *
1569         *
1570         *
1571         *
1572         *
1573         *
1574         *
1575         *
1576         *
1577         *
1578         *
1579         *
1580         *
1581         *
1582         *
1583         *
1584         *
1585         *
1586         *
1587         *
1588         *
1589         *
1590         *
1591         *
1592         *
1593         *
1594         *
1595         *
1596         *
1597         *
1598         *
1599         *
1600         *
1601         *
1602         *
1603         *
1604         *
1605         *
1606         *
1607         *
1608         *
1609         *
1610         *
1611         *
1612         *
1613         *
1614         *
1615         *
1616         *
1617         *
1618         *
1619         *
1620         *
1621         *
1622         *
1623         *
1624         *
1625         *
1626         *
1627         *
1628         *
1629         *
1630         *
1631         *
1632         *
1633         *
1634         *
1635         *
1636         *
1637         *
1638         *
1639         *
1640         *
1641         *
1642         *
1643         *
1644         *
1645         *
1646         *
1647         *
1648         *
1649         *
1650         *
1651         *
1652         *
1653         *
1654         *
1655         *
1656         *
1657         *
1658         *
1659         *
1660         *
1661         *
1662         *
1663         *
1664         *
1665         *
1666         *
1667         *
1668         *
1669         *
1670         *
1671         *
1672         *
1673         *
1674         *
1675         *
1676         *
1677         *
1678         *
1679         *
1680         *
1681         *
1682         *
1683         *
1684         *
1685         *
1686         *
1687         *
1688         *
1689         *
1690         *
1691         *
1692         *
1693         *
1694         *
1695         *
1696         *
1697         *
1698         *
1699         *
1700         *
1701         *
1702         *
1703         *
1704         *
1705         *
1706         *
1707         *
1708         *
1709         *
1710         *
1711         *
1712         *
1713         *
1714         *
1715         *
1716         *
1717         *
1718         *
1719         *
1720         *
1721         *
1722         *
1723         *
1724         *
1725         *
1726         *
1727         *
1728         *
1729         *
1730         *
1731         *
1732         *
1733         *
1734         *
1735         *
1736         *
1737         *
1738         *
1739         *
1740         *
1741         *
1742         *
1743         *
1744         *
1745         *
1746         *
1747         *
1748         *
1749         *
1750         *
1751         *
1752         *
1753         *
1754         *
1755         *
1756         *
1757         *
1758         *
1759         *
1760         *
1761         *
1762         *
1763         *
1764         *
1765         *
1766         *
1767         *
1768         *
1769         *
1770         *
1771         *
1772         *
1773         *
1774         *
1775         *
1776         *
1777         *
1778         *
1779         *
1780         *
1781         *
1782         *
1783         *
1784         *
1785         *
1786         *
1787         *
1788         *
1789         *
1790         *
1791         *
1792         *
1793         *
1794         *
1795         *
1796         *
1797         *
1798         *
1799         *
1800         *
1801         *
1802         *
1803         *
1804         *
1805         *
1806         *
1807         *
1808         *
1809         *
1810         *
1811         *
1812         *
1813         *
1814         *
1815         *
1816         *
1817         *
1818         *
1819         *
1820         *
1821         *
1822         *
1823         *
1824         *
1825         *
1826         *
1827         *
1828         *
1829         *
1830         *
1831         *
1832         *
1833         *
1834         *
1835         *
1836         *
1837         *
1838         *
1839         *
1840         *
1841         *
1842         *
1843         *
1844         *
1845         *
1846         *
1847         *
1848         *
1849         *
1850         *
1851         *
1852         *
1853         *
1854         *
1855         *
1856         *
1857         *
1858         *
1859         *
1860         *
1861         *
1862         *
1863         *
1864         *
1865         *
1866         *
1867         *
1868         *
1869         *
1870         *
1871         *
1872         *
1873         *
1874         *
1875         *
1876         *
1877         *
1878         *
1879         *
1880         *
1881         *
1882         *
1883         *
1884         *
1885         *
1886         *
1887         *
1888         *
1889         *
1890         *
1891         *
1892         *
1893         *
1894         *
1895         *
1896         *
1897         *
1898         *
1899         *
1900         *
1901         *
1902         *
1903         *
1904         *
1905         *
1906         *
1907         *
1908         *
1909         *
1910         *
1911         *
1912         *
1913         *
1914         *
1915         *
1916         *
1917         *
1918         *
1919         *
1920         *
1921         *
1922         *
1923         *
1924         *
1925         *
1926         *
1927         *
1928         *
1929         *
1930         *
1931         *
1932         *
1933         *
1934         *
1935         *
1936         *
1937         *
1938         *
1939         *
1940         *
1941         *
1942         *
1943         *
1944         *
1945         *
1946         *
1947         *
1948         *
1949         *
1950         *
1951         *
1952         *
1953         *
1954         *
1955         *
1956         *
1957         *
1958         *
1959         *
1960         *
1961         *
1962         *
1963         *
1964         *
1965         *
1966         *
1967         *
1968         *
1969         *
1970         *
1971         *
1972         *
1973         *
1974         *
1975         *
1976         *
1977         *
1978         *
1979         *
1980         *
1981         *
1982         *
1983         *
1984         *
1985         *
1986         *
1987         *
1988         *
1989         *
1990         *
1991         *
1992         *
1993         *
1994         *
1995         *
1996         *
1997         *
1998         *
1999         *
2000         *
2001         *
2002         *
2003         *
2004         *
2005         *
2006         *
2007         *
2008         *
2009         *
2010         *
2011         *
2012         *
2013         *
2014         *
2015         *
2016         *
2017         *
2018         *
2019         *
2020         *
2021         *
2022         *
2023         *
2024         *
2025         *
2026         *
2027         *
2028         *
2029         *
2030         *
2031         *
2032         *
2033         *
2034         *
2035         *
2036         *
2037         *
2038         *
2039         *
2040         *
2041         *
2042         *
2043         *
2044         *
2045         *
2046         *
2047         *
2048         *
2049         *
2050         *
2051         *
2052         *
2053         *
2054         *
2055         *
2056         *
2057         *
2058         *
2059         *
2060         *
2061         *
2062         *
2063         *
2064         *
2065         *
2066         *
2067         *
2068         *
2069         *
2070         *
2071         *
2072         *
2073         *
2074         *
2075         *
2076         *
2077         *
2078         *
2079         *
2080         *
2081         *
2082         *
2083         *
2084         *
2085         *
2086         *
2087         *
2088         *
2089         *
2090         *
2091         *
2092         *
2093         *
2094         *
2095         *
2096         *
2097         *
2098         *
2099         *
2100         *
2101         *
2102         *
2103         *
2104         *
2105         *
2106         *
2107         *
2108         *
2109         *
2110         *
2111         *
2112         *
2113         *
2114         *
2115         *
2116         *
2117         *
2118         *
2119         *
2120         *
2121         *
2122         *
2123         *
2124         *
2125         *
2126         *
2127         *
2128         *
2129         *
2130         *
2131         *
2132         *
2133         *
2134         *
2135         *
2136         *
2137         *
2138         *
2139         *
2140         *
2141         *
2142         *
2143         *
2144         *
2145         *
2146         *
2147         *
2148         *
2149         *
2150         *
2151         *
2152         *
2153         *
2154         *
2155         *
2156         *
2157         *
2158         *
2159         *
2160         *
2161         *
2162         *
2163         *
2164         *
2165         *
2166         *
2167         *
2168         *
2169         *
2170         *
2171         *
2172         *
2173         *
2174         *
2175         *
2176         *
2177         *
2178         *
2179         *
2180         *
2181         *
2182         *
2183         *
2184         *
2185         *
2186         *
2187         *
2188         *
2189         *
2190         *
2191         *
2192         *
2193         *
2194         *
2195         *
2196         *
2197         *
2198         *
2199         *
2200         *
2201         *
2202         *
2203         *
2204         *
2205         *
2206         *
2207         *
2208         *
2209         *
2210         *
2211         *
2212         *
2213         *
2214         *
2215         *
2216         *
2217         *
2218         *
2219         *
2220         *
2221         *
2222         *
2223         *
2224         *
2225         *
2226         *
2227         *
2228         *
2229         *
2230         *
2231         *
2232         *
2233         *
2234         *
2235         *
2236         *
2237         *
2238         *
2239         *
2240         *
2241         *
2242         *
2243         *
2244         *
2245         *
2246         *
2247         *
2248         *
2249         *
2250         *
2251         *
2252         *
2253         *
2254         *
2255         *
2256         *
2257         *
2258         *
2259         *
2260         *
2261         *
2262         *
2263         *
2264         *
2265         *
2266         *
2267         *
2268         *
2269         *
2270         *
2271         *
2272         *
2273         *
2274         *
2275         *
2276         *
2277         *
2278         *
2279         *
2280         *
2281         *
2282         *
2283         *
2284         *
2285         *
2286         *
2287         *
2288         *
2289         *
2290         *
2291         *
2292         *
2293         *
2294         *
2295         *
2296         *
2297         *
2298         *
2299         *
2300         *
2301         *
2302         *
2303         *
2304         *
2305         *
2306         *
2307         *
2308         *
2309         *
2310         *
2311         *
2312         *
2313         *
2314         *
2315         *
2316         *
2317         *
2318         *
2319         *
2320         *
2321         *
2322         *
2323         *
2324         *
2325         *
2326         *
2327         *
2328         *
2329         *
2330         *
2331         *
2332         *
2333         *
2334         *
2335         *
2336         *
2337         *
2338         *
2339         *
2340         *
2341         *
2342         *
2343         *
2344         *
2345         *
2346         *
2347         *
2348         *
2349         *
2350         *
2351         *
2352         *
2353         *
2354         *
2355         *
2356         *
2357         *
2358         *
2359         *
2360         *
2361         *
2362         *
2363         *
2364         *
2365         *
2366         *
2367         *
2368         *
2369         *
2370         *
2371         *
2372         *
2373         *
2374         *
2375         *
2376         *
2377         *
2378         *
2379         *
2380         *
2381         *
2382         *
2383         *
2384         *
2385         *
2386         *
2387         *
2388         *
2389         *
2390         *
2391         *
2392         *
2393         *
2394         *
2395         *
2396         *
2397         *
2398         *
2399         *
2400         *
2401         *
2402         *
2403         *
2404         *
2405         *
2406         *
2407         *
2408         *
2409         *
2410         *
2411         *
2412         *
2413         *
2414         *
2415         *
2416         *
2417         *
2418         *
2419         *
2420         *
2421         *
2422         *
2423         *
2424         *
2425         *
2426         *
2427         *
2428         *
2429         *
2430         *
2431         *
2432         *
2433         *
2434         *
2435         *
2436         *
2437         *
2438         *
2439         *
2440         *
2441         *
2442         *
2443         *
2444         *
2445         *
2446         *
2447         *
2448         *
2449         *
2450         *
2451         *
2452         *
2453         *
2454         *
2455         *
2456         *
2457         *
2458         *
2459         *
2460         *
2461         *
2462         *
2463         *
2464         *
2465         *
2466         *
2467         *
2468         *
2469         *
2470         *
2471         *
2472         *
2473         *
2474         *
2475         *
2476         *
2477         *
2478         *
2479         *
2480         *
2481         *
2482         *
2483         *
2484         *
2485         *
2486         *
2487         *
2488         *
2489         *
2490         *
2491         *
2492         *
2493         *
2494         *
2495         *
2496         *
2497         *
2498         *
2499         *
2500         *
2501         *
2502         *
2503         *
2504         *
2505         *
2506         *
2507         *
2508         *
2509         *
2510         *
2511         *
2512         *
2513         *
2514         *
2515         *
2516         *
2517         *
2518         *
2519         *
2520         *
2521         *
2522         *
2523         *
2524         *
2525         *
2526         *
2527         *
2528         *
2529         *
2530         *
2531         *
2532         *
2533         *
2534         *
2535         *
2536         *
2537         *
2538         *
2539         *
2540         *
2541         *
2542         *
2543         *
2544         *
2545         *
2546         *
2547         *
2548         *
2549         *
2550         *
2551         *
2552         *
2553         *
2554         *
2555         *
2556         *
2557         *
2558         *
2559         *
2560         *
2561         *
2562         *
2563         *
2564         *
2565         *
2566         *
2567         *
2568         *
2569         *
2570         *
2571         *
2572         *
2573         *
2574         *
2575         *
2576         *
2577         *
2578         *
2579         *
2580         *
2581         *
2582         *
2583         *
2584         *
2585         *
2586         *
2587         *
2588         *
2589         *
2590         *
2591         *
2592         *
2593         *
2594         *
2595         *
2596         *
2597         *
2598         *
2599         *
2600         *
2601         *
2602         *
2603         *
2604         *
2605         *
2606         *
2607         *
2608         *
2609         *
2610         *
2611         *
2612         *
2613         *
2614         *
2615         *
2616         *
2617         *
2618         *
2619        
```



```

1225         errbuf);
1226         goto error;
1227     }
1228     break;

1230     case ZFS_PROP_VOLSIZE:
1231         if (intval % blocksize != 0) {
1232             zfs_nicenum(blocksize, buf,
1233                 sizeof (buf));
1234             zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
1235                 "%s' must be a multiple of "
1236                 "volume block size (%s)"),
1237                 propname, buf);
1238             (void) zfs_error(hdl, EZFS_BADPROP,
1239                 errbuf);
1240             goto error;
1241         }

1243         if (intval == 0) {
1244             zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
1245                 "%s' cannot be zero"),
1246                 propname);
1247             (void) zfs_error(hdl, EZFS_BADPROP,
1248                 errbuf);
1249             goto error;
1250         }
1251     }
1252     break;
1253 }
1254 }

1256 /*
1257  * If normalization was chosen, but no UTF8 choice was made,
1258  * enforce rejection of non-UTF8 names.
1259  *
1260  * If normalization was chosen, but rejecting non-UTF8 names
1261  * was explicitly not chosen, it is an error.
1262  */
1263 if (chosen_normal > 0 && chosen_utf < 0) {
1264     if (nvlist_add_uint64(ret,
1265         zfs_prop_to_name(ZFS_PROP_UTF8ONLY), 1) != 0) {
1266         (void) no_memory(hdl);
1267         goto error;
1268     }
1269 } else if (chosen_normal > 0 && chosen_utf == 0) {
1270     zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
1271         "%s' must be set 'on' if normalization chosen"),
1272         zfs_prop_to_name(ZFS_PROP_UTF8ONLY));
1273     (void) zfs_error(hdl, EZFS_BADPROP, errbuf);
1274     goto error;
1275 }
1276 return (ret);

1278 error:
1279     nvlist_free(ret);
1280     return (NULL);
1281 }

1283 int
1284 zfs_add_synthetic_resv(zfs_handle_t *zhp, nvlist_t *nvl)
1285 {
1286     uint64_t old_volsize;
1287     uint64_t new_volsize;
1288     uint64_t old_reservation;
1289     uint64_t new_reservation;
1290     zfs_prop_t resv_prop;

```

```

1291     uint64_t volblocksize;
1292     int ncopies;
1293     nvlist_t *props;

1294     /*
1295      * If this is an existing volume, and someone is setting the volsize,
1296      * make sure that it matches the reservation, or add it if necessary.
1297      */
1298     old_volsize = zfs_prop_get_int(zhp, ZFS_PROP_VOLSIZE);
1299     if (zfs_which_resv_prop(zhp, &resv_prop) < 0)
1300         return (-1);
1301     old_reservation = zfs_prop_get_int(zhp, resv_prop);
1302     volblocksize = zfs_prop_get_int(zhp, ZFS_PROP_VOLBLOCKSIZE);
1303     ncopies = zfs_prop_get_int(zhp, ZFS_PROP_COPIES);

1305     if ((zvol_volsize_to_reservation(old_volsize, volblocksize,
1306         ncopies) != old_reservation) || nvlist_exists(nvl,
1307         zfs_prop_to_name(resv_prop)))
1308         props = nvlist_alloc();
1309     nvlist_add_uint64(props, zfs_prop_to_name(ZFS_PROP_VOLBLOCKSIZE),
1310         zfs_prop_get_int(zhp, ZFS_PROP_VOLBLOCKSIZE));

1312     if ((zvol_volsize_to_reservation(old_volsize, props) !=
1313         old_reservation) || nvlist_exists(nvl,
1314         zfs_prop_to_name(resv_prop))) {
1315         nvlist_free(props);
1316         return (0);
1317     }

1319     if (nvlist_lookup_uint64(nvl, zfs_prop_to_name(ZFS_PROP_VOLSIZE),
1320         &new_volsize) != 0)
1321         &new_volsize != 0) {
1322         nvlist_free(props);
1323         return (-1);
1324     }
1325     new_reservation = zvol_volsize_to_reservation(new_volsize,
1326         volblocksize, ncopies);
1327     new_reservation = zvol_volsize_to_reservation(new_volsize, props);
1328     nvlist_free(props);

1330     if (nvlist_add_uint64(nvl, zfs_prop_to_name(resv_prop),
1331         new_reservation) != 0) {
1332         (void) no_memory(zhp->zfs_hdl);
1333         return (-1);
1334     }
1335     return (1);
1336 }

1338 unchanged portion omitted

4460 /*
4461  * Computes the required reservation to completely contain all blocks of a
4462  * zvol at a given volsize.
4463  * Convert the zvol's volume size to an appropriate reservation.
4464  * Note: If this routine is updated, it is necessary to update the ZFS test
4465  * suite's shell version in reservation.kshlib.
4466  */
4467 uint64_t
4468 zvol_volsize_to_reservation(uint64_t volsize, uint64_t volblocksize,
4469     int ncopies)
4470 {
4471     uint64_t numdb;
4472     uint64_t nblocks;
4473     uint64_t nblocks, volblocksize;
4474     int ncopies;
4475     char *strval;

```

```
4475     if (nvlist_lookup_string(props,
4476         zfs_prop_to_name(ZFS_PROP_COPIES), &strval) == 0)
4477         ncopies = atoi(strval);
4478     else
4479         ncopies = 1;
4480     if (nvlist_lookup_uint64(props,
4481         zfs_prop_to_name(ZFS_PROP_VOLBLOCKSIZE),
4482         &volblocksize) != 0)
4483         volblocksize = ZVOL_DEFAULT_BLOCKSIZE;
4471     nblocks = volsize/volblocksize;
4472     /* start with metadnode L0-L6 */
4473     numdb = 7;
4474     /* calculate number of indirects */
4475     while (nblocks > 1) {
4476         nblocks += DNODES_PER_LEVEL - 1;
4477         nblocks /= DNODES_PER_LEVEL;
4478         numdb += nblocks;
4479     }
4480     numdb *= MIN(SPA_DVAS_PER_BP, ncopies + 1);
4481     volsize *= ncopies;
4482     /*
4483      * this is exactly DN_MAX_INDBLKSHIFT when metadata isn't
4484      * compressed, but in practice they compress down to about
4485      * 1100 bytes
4486      */
4487     numdb *= 1ULL << DN_MAX_INDBLKSHIFT;
4488     volsize += numdb;
4489     return (volsize);
4490 }
```

unchanged portion omitted