

new/usr/src/cmd/zfs/zfs_main.c

```
new/usr/src/cmd/zfs/zfs_main.c

731                                     goto error;
732
733     }
734
735     if (nvlist_add_uint64(props,
736         zfs_prop_to_name(ZFS_PROP_VOLBLOCKSIZE),
737         intval) != 0)
738         nomem();
739     break;
740 case 'o':
741     if (parseprop(props))
742         goto error;
743     break;
744 case 's':
745     noreserve = B_TRUE;
746     break;
747 case ':':
748     (void) fprintf(stderr, gettext("missing size "
749                     "argument\n"));
750     goto badusage;
751 case '?':
752     (void) fprintf(stderr, gettext("invalid option '%c'\n"),
753                     optarg);
754     goto badusage;
755 }

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

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

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

775 if (type == ZFS_TYPE_VOLUME && !noreserve) {
776     zpool_handle_t *zpool_handle;
777     nvlist_t *real_props;
778     uint64_t spa_version;
779     char *p;
780     zfs_prop_t resv_prop;
781     char *strval;
782     char msg[1024];
783     uint64_t volblocksize;
784     int ncopies;
785
786     if (p = strchr(argv[0], '/'))
787         *p = '\0';
788     zpool_handle = zpool_open(g_zfs, argv[0]);
789     if (p != NULL)
790         *p = '/';
791     if (zpool_handle == NULL)
792         goto error;
793     spa_version = zpool_get_prop_int(zpool_handle,
794         ZPOOL_PROP_VERSION, NULL);
795 }
```

```

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) sprintf(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_impl(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

new/usr/src/lib/libzfs/common/libzfs.h

1

```
*****
27130 Tue Oct 15 13:59:52 2013
new/usr/src/lib/libzfs/common/libzfs.h
4012 Upper limit of zfs set bounds check for refreservation 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 */
```

new/usr/src/lib/libzfs/common/libzfs.h

2

```
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_CROSSTARGET,     /* rename or clone across pool or dataset */  
87     EZFS_ZONEDE,          /* used improperly in local zone */  
88     EZFS_MOUNTFAILED,     /* failed to mount dataset */  
89     EZFS_UNMOUNTFAILED,   /* failed to umount dataset */  
90     EZFS_UNSHARENSFAILED,/* unshare(1M) failed */  
91     EZFS_SHARENSFAILED,   /* 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_INVALCONFIG,      /* 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_INVALARG,    /* 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_RELEASE,   /* 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_nv1(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, nvlist_t *);
604 extern uint64_t zvol_volsize_to_reservation_impl(uint64_t volsize,
605     uint64_t volblocksize, int ncopies);

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

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

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

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

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

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

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

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

635     /* set "camount=off" on all modified filesystems */
636     boolean_t camountoff;

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

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


---

unchanged portion omitted
```

new/usr/src/lib/libzfs/common/libzfs_dataset.c

```
*****
112221 Tue Oct 15 13:59:52 2013
new/usr/src/lib/libzfs/common/libzfs_dataset.c
4012 Upper limit of zfs set bounds check for refreservation 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 */

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"

1

new/usr/src/lib/libzfs/common/libzfs_dataset.c

```
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_INVAL && 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     }  
833     if (nvlist_free(ret) != 0)  
834         (void) no_memory(hdl);  
835     return (ret);  
836 }  
837  
838 error:  
839     if (errbuf != NULL)  
840         (void) no_memory(hdl);  
841     return (NULL);  
842 }
```

2

```
new/usr/src/lib/libzfs/common/libzfs_dataset.c

830                                         }
832                                         if (strlen(nvpair_name) > 0)
833                                         zfs_error_aux(hdl, "property %s does not exist", nvpair_name);
834                                         propname = nvpair_name;
835                                         propname_len = strlen(propname);
836                                         (void) zfs_error_aux(hdl, "property %s does not exist", propname);
837                                         goto error;
838                                     }
839
840                                     (void) nvpair_value;
841                                     if (nvlist_add_string_nvpair(nvlist, nvpair_name, nvpair_value) != 0)
842                                         (void) no_error;
843                                         goto error;
844                                     }
845                                     continue;
846                                 }
847
848                                 /*
849                                 * Currently, only user properties
850                                 * snapshots.
851                                 */
852                                 if (type == ZFS_TYPE_SNAPSHOT)
853                                     zfs_error_aux(hdl, "this property is not supported for snapshots");
854                                     (void) zfs_error_aux(hdl, "property %s does not exist", propname);
855                                     goto error;
856                                 }
857
858                                 if (prop == ZPROP_INVAL & !is_userquota_property)
859                                     zfs_userquota_property_error(hdl, propname, domain, rid, valary);
860
861                                     if (userquota_property_error(hdl, propname, domain, rid, valary))
862                                         char newpropname[PROPNAME_MAX];
863                                         char domain[128];
864                                         uint64_t rid;
865                                         uint64_t valary[3];
866
867                                         if (userquota_property_error(hdl, newpropname, domain, rid, valary))
868                                             &uqtype, domain);
869                                         zfs_error_aux(hdl, "property %s does not exist", newpropname);
870                                         dgettree(domain);
871                                         "'%s'";
872                                         propname = newpropname;
873                                         (void) zfs_error_aux(hdl, "property %s does not exist", propname);
874                                         goto error;
875
876                                 if (uqtype != ZFS_PROP_SNAPSHOT & uqtype != ZFS_PROP_SNAPSHOT)
877                                     zfs_error_aux(hdl, "property %s does not support snapshot propagation", propname);
878                                     (void) zfs_error_aux(hdl, "property %s does not support snapshot propagation", propname);
879                                     errbuf[0] = '\0';
880                                     goto error;
881
882                                 if (nvpair_type(e) == NVP_TYPE_NVPAIR)
883                                     if ((void) nvpair_value(e) != 0)
884                                         goto error;
885
886                                 if (nvpair_type(e) == NVP_TYPE_NVPAIR)
887                                     (void) nvpair_value(e);
888                                     if (strcmp(strval, "true") == 0)
889                                         in = 1;
890                                     } else if (strcmp(strval, "false") == 0)
891                                         in = 0;
892                                     strval = (char *)strchr(strval, '\0');
893                                     if (*strval != '\0')
894                                         goto error;
895                                 }
```

```

new/usr/src/lib/libzfs/common/libzfs_dataset.c

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) sprintf(newpropname, sizeof (newpropname),
918                             "%s%llx-%s",
919                             zfs_userquota_prop_prefixes[uqtype],
920                             (longlong_t)rid, domain);
921             valary[0] = uqtype;
922             valary[1] = rid;
923             valary[2] = intval;
924             if (nvlist_add_uint64_array(ret, newpropname,
925                             valary, 3) != 0) {
926                 (void) no_memory(hdl);
927                 goto error;
928             }
929             continue;
930         } else if (prop == ZPROP_INVAL && zfs_prop_written(propname)) {
931             zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
932                             "'%s' is readonly"),
933                             propname);
934             (void) zfs_error(hdl, EZFS_PROP_READONLY, errbuf);
935             goto error;
936         }
937         if (prop == ZPROP_INVAL) {
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         if (!zfs_prop_valid_for_type(prop, type)) {
944             zfs_error_aux(hdl,
945                             dgettext(TEXT_DOMAIN, "'%s' does not "
946                             "apply to datasets of this type"), propname);
947             (void) zfs_error(hdl, EZFS_PROPTYPE, errbuf);
948             goto error;
949         }
950
951         if (zfs_prop_READONLY(prop) &&
952             (!zfs_prop_Setone(prop) || zhp != NULL)) {
953             zfs_error_aux(hdl,
954                             dgettext(TEXT_DOMAIN, "'%s' is readonly"),
955                             propname);
956             (void) zfs_error(hdl, EZFS_PROP_READONLY, errbuf);
957             goto error;
958         }
959
960         if (zprop_parse_value(hdl, elem, prop, type, ret,

```

```

962             &strval, &intval, errbuf) != 0)
963             goto error;
964
965         /* Perform some additional checks for specific properties.
966         */
967         switch (prop) {
968             case ZFS_PROP_VERSION:
969             {
970                 int version;
971
972                 if (zhp == NULL)
973                     break;
974                 version = zfs_prop_get_int(zhp, ZFS_PROP_VERSION);
975                 if (intval < version) {
976                     zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
977                         "Can not downgrade; already at version %u"),
978                         version);
979                     (void) zfs_error(hdl, EZFS_BADPROP, errbuf);
980                     goto error;
981                 }
982                 break;
983             }
984
985             case ZFS_PROP_RECORDSIZE:
986             case ZFS_PROP_VOLBLOCKSIZE:
987                 /* must be power of two within SPA_{MIN,MAX}BLOCKSIZE */
988                 if (intval < SPA_MINBLOCKSIZE ||
989                     intval > SPA_MAXBLOCKSIZE || !ISP2(intval)) {
990                     zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
991                         "'%s' must be power of 2 from %u "
992                         "to %u"), propname,
993                         (uint_t)SPA_MINBLOCKSIZE,
994                         (uint_t)SPA_MAXBLOCKSIZE >> 10);
995                     (void) zfs_error(hdl, EZFS_BADPROP, errbuf);
996                     goto error;
997                 }
998                 break;
999
1000            case ZFS_PROP_MSLABEL:
1001            {
1002                /*
1003                 * Verify the mslabel string and convert to
1004                 * internal hex label string.
1005                */
1006
1007                m_label_t *new_sl;
1008                char *hex; /* internal label string */
1009
1010                /* Default value is already OK. */
1011                if (strcasecmp(strval, ZFS_MSLABEL_DEFAULT) == 0)
1012                    break;
1013
1014                /* Verify the label can be converted to binary form */
1015                if (((new_sl = m_label_alloc(MAC_LABEL)) == NULL) ||
1016                    (str_to_label(strval, &new_sl, MAC_LABEL,
1017                        L_NO_CORRECTION, NULL) == -1)) {
1018                    goto badlabel;
1019                }
1020
1021                /* Now translate to hex internal label string */
1022                if (label_to_str(new_sl, &hex, M_INTERNAL,
1023                    DEF_NAMES) != 0) {
1024                    if (hex)
1025                        free(hex);
1026                    goto badlabel;
1027

```

```

1028             }
1029             m_label_free(new_sl);
1030
1031         /* If string is already in internal form, we're done. */
1032         if (strcmp(strval, hex) == 0) {
1033             free(hex);
1034             break;
1035         }
1036
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);
1043
1044     break;
1045
1046     badlabel:
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
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
2099
2100
2100
2101
2101
2102
2102
2103
2103
2104
2104
2105
2105
2106
2106
2107
2107
2108
2108
2109
2109
2110
2110
2111
2111
2112
2112
2113
2113
2114
2114
2115
2115
2116
2116
2117
2117
2118
2118
2119
2119
2120
2120
2121
2121
2122
2122
2123
2123
2124
2124
2125
2125
2126
2126
2127
2127
2128
2128
2129
2129
2130
2130
2131
2131
2132
2132
2133
2133
2134
2134
2135
2135
2136
2136
2137
2137
2138
2138
2139
2139
2140
2140
2141
2141
2142
2142
2143
2143
2144
2144
2145
2145
2146
2146
2147
2147
2148
2148
2149
2149
2150
2150
2151
2151
2152
2152
2153
2153
2154
2154
2155
2155
2156
2156
2157
2157
2158
2158
2159
2159
2160
2160
2161
2161
2162
2162
2163
2163
2164
2164
2165
2165
2166
2166
2167
2167
2168
2168
2169
2169
2170
2170
2171
2171
2172
2172
2173
2173
2174
2174
2175
2175
2176
2176
2177
2177
2178
2178
2179
2179
2180
2180
2181
2181
2182
2182
2183
2183
2184
2184
2185
2185
2186
2186
2187
2187
2188
2188
2189
2189
2190
2190
2191
2191
2192
2192
2193
2193
2194
2194
2195
2195
2196
2196
2197
2197
2198
2198
2199
2199
2200
2200
2201
2201
2202
2202
2203
2203
2204
2204
2205
2205
2206
2206
2207
2207
2208
2208
2209
2209
2210
2210
2211
2211
2212
2212
2213
2213
2214
2214
2215
2215
2216
2216
2217
2217
2218
2218
2219
2219
2220
2220
2221
2221
2222
2222
2223
2223
2224
2224
2225
2225
2226
2226
2227
2227
2228
2228
2229
2229
2230
2230
2231
2231
2232
2232
2233
2233
2234
2234
2235
2235
2236
2236
2237
2237
2238
2238
2239
2239
2240
2240
2241
2241
2242
2242
2243
2243
2244
2244
2245
2245
2246
2246
2247
2247
2248
2248
2249
2249
2250
2250
2251
2251
2252
2252
2253
2253
2254
2254
2255
2255
2256
2256
2257
2257
2258
2258
2259
2259
2260
2260
2261
2261
2262
2262
2263
2263
2264
2264
2265
2265
2266
2266
2267
2267
2268
2268
2269
2269
2270
2270
2271
2271
2272
2272
2273
2273
2274
2274
2275
2275
2276
2276
2277
2277
2278
2278
2279
2279
2280
2280
2281
2281
2282
2282
2283
2283
2284
2284
2285
2285
2286
2286
2287
2287
2288
2288
2289
2289
2290
2290
2291
2291
2292
2292
2293
2293
2294
2294
2295
2295
2296
2296
2297
2297
2298
2298
2299
2299
2300
2300
2301
2301
2302
2302
2303
2303
2304
2304
2305
2305
2306
2306
2307
2307
2308
2308
2309
2309
2310
2310
2311
2311
2312
2312
2313
2313
2314
2314
2315
2315
2316
2316
2317
2317
2318
2318
2319
2319
2320
2320
2321
2321
2322
2322
2323
2323
2324
2324
2325
2325
2326
2326
2327
2327
2328
2328
2329
2329
2330
2330
2331
2331
2332
2332
2333
2333
2334
2334
2335
2335
2336
2336
2337
2337
2338
2338
2339
2339
2340
2340
2341
2341
2342
2342
2343
2343
2344
2344
2345
2345
2346
2346
2347
2347
2348
2348
2349
2349
2350
2350
2351
2351
2352
2352
2353
2353
2354
2354
2355
2355
2356
2356
2357
2357
2358
2358
2359
2359
2360
2360
2361
2361
2362
2362
2363
2363
2364
2364
2365
2365
2366
2366
2367
2367
2368
2368
2369
2369
2370
2370
2371
2371
2372
2372
2373
2373
2374
2374
2375
2375
2376
2376
2377
2377
2378
2378
2379
2379
2380
2380
2381
2381
2382
2382
2383
2383
2384
2384
2385
2385
2386
2386
2387
2387
2388
2388
2389
2389
2390
2390
2391
2391
2392
2392
2393
2393
2394
2394
2395
2395
2396
2396
2397
2397
2398
2398
2399
2399
2400
2400
2401
2401
2402
2402
2403
2403
2404
2404
2405
2405
2406
2406
2407
2407
2408
2408
2409
2409
2410
2410
2411
2411
2412
2412
2413
2413
2414
2414
2415
2415
2416
2416
2417
2417
2418
2418
2419
2419
2420
2420
2421
2421
2422
2422
2423
2423
2424
2424
2425
2425
2426
2426
2427
2427
2428
2428
2429
2429
2430
2430
2431
2431
2432
2432
2433
2433
2434
2434
2435
2435
2436
2436
2437
2437
2438
2438
2439
2439
2440
2440
2441
2441
2442
2442
2443
2443
2444
2444
2445
2445
2446
2446
2447
2447
2448
2448
2449
2449
2450
2450
2451
2451
2452
2452
2453
2453
2454
2454
2455
2455
2456
2456
2457
2457
2458
2458
2459
2459
2460
2460
2461
2461
2462
2462
2463
2463
2464
2464
2465
2465
2466
2466
2467
2467
2468
2468
2469
2469
2470
2470
2471
2471
2472
2472
2473
2473
2474
2474
2475
2475
2476
2476
2477
2477
2478
2478
2479
2479
2480
2480
2481
2481
2482
2482
2483
2483
2484
2484
2485
2485
2486
2486
2487
2487
2488
2488
2489
2489
2490
2490
2491
2491
2492
2492
2493
2493
2494
2494
2495
2495
2496
2496
2497
2497
2498
2498
2499
2499
2500
2500
2501
2501
2502
2502
2503
2503
2504
2504
2505
2505
2506
2506
2507
2507
2508
2508
2509
2509
2510
2510
2511
2511
2512
2512
2513
2513
2514
2514
2515
2515
2516
2516
2517
2517
2518
2518
2519
2519
2520
2520
2521
2521
2522
2522
2523
2523
2524
2524
2525
2525
2526
2526
2527
2527
2528
2528
2529
2529
2530
2530
2531
2531
2532
2532
2533
2533
2534
2534
2535
2535
2536
2536
2537
2537
2538
2538
2539
2539
2540
2540
2541
2541
2542
2542
2543
2543
2544
2544
2545
2545
2546
2546
2547
2547
2548
2548
2549
2549
2550
2550
2551
2551
2552
2552
2553
2553
2554
2554
2555
2555
2556
2556
2557
2557
2558
2558
2559
2559
2560
2560
2561
2561
2562
2562
2563
2563
2564
2564
2565
2565
2566
2566
2567
2567
2568
2568
2569
2569
2570
2570
2571
2571
2572
2572
2573
2573
2574
2574
2575
2575
2576
2576
2577
2577
2578
2578
2579
2579
2580
2580
2581
2581
2582
2582
2583
2583
2584
2584
2585
2585
2586
2586
2587
2587
2588
2588
2589
2589
2590
2590
2591
2591
2592
2592
2593
2593
2594
2594
2595
2595
2596
2596
2597
2597
```

new/usr/src/lib/libzfs/common/libzfs_dataset.c

7

```

1094 * -----
1095 * zoned=on      mountpoint (no)      mountpoint (yes)
1096 *          sharenfs (no)        sharenfs (no)
1097 *          sharesmb (no)        sharesmb (no)
1098 *
1099 * zoned=off      mountpoint (yes)      N/A
1100 *          sharenfs (yes)
1101 *          sharesmb (yes)
1102 */
1103 if (zoned) {
1104     if (getzoneid() == GLOBAL_ZONEID) {
1105         zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
1106             "'%s' cannot be set on "
1107             "dataset in a non-global zone"),
1108             propname);
1109         (void) zfs_error(hdl, EZFS_ZONED,
1110             errbuf);
1111         goto error;
1112     } else if (prop == ZFS_PROP_SHARENFS ||

1113         prop == ZFS_PROP_SHARESMB) {
1114         zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
1115             "'%s' cannot be set in "
1116             "a non-global zone"), propname);
1117         (void) zfs_error(hdl, EZFS_ZONED,
1118             errbuf);
1119         goto error;
1120     }
1121 } else if (getzoneid() != GLOBAL_ZONEID) {
1122     /*
1123     * If zoned property is 'off', this must be in
1124     * a global zone. If not, something is wrong.
1125     */
1126     zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
1127         "'%s' cannot be set while dataset "
1128         "'zoned' property is set"), propname);
1129     (void) zfs_error(hdl, EZFS_ZONED, errbuf);
1130     goto error;
1131 }

1132 /*
1133 * At this point, it is legitimate to set the
1134 * property. Now we want to make sure that the
1135 * property value is valid if it is sharenfs.
1136 */
1137 if ((prop == ZFS_PROP_SHARENFS ||
1138     prop == ZFS_PROP_SHARESMB) &&
1139     strcmp(strval, "on") != 0 &&
1140     strcmp(strval, "off") != 0) {
1141     zfs_share_proto_t proto;

1142     if (prop == ZFS_PROP_SHARESMB)
1143         proto = PROTO_SMB;
1144     else
1145         proto = PROTO_NFS;

1146     /*
1147     * Must be an valid sharing protocol
1148     * option string so init the libshare
1149     * in order to enable the parser and
1150     * then parse the options. We use the
1151     * control API since we don't care about
1152     * the current configuration and don't
1153     * want the overhead of loading it
1154     * until we actually do something.
1155     */
1156
1157
1158

```

new/usr/src/lib/libzfs/common/libzfs_dataset.c

```

if (zfs_init_libshare(hdl,
    SA_INIT_CONTROL_API) != SA_OK) {
    /*
     * An error occurred so we can't do
     * anything
     */
    zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
        "'%s' cannot be set: problem "
        "in share initialization"),
        propname);
    (void) zfs_error(hdl, EZFS_BADPROP,
        errbuf);
    goto error;
}

if (zfs_parse_options(strval, proto) != SA_OK) {
    /*
     * There was an error in parsing so
     * deal with it by issuing an error
     * message and leaving after
     * uninitialized the libshare
     * interface.
     */
    zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
        "'%s' cannot be set to invalid "
        "options"), propname);
    (void) zfs_error(hdl, EZFS_BADPROP,
        errbuf);
    zfs_uninit_libshare(hdl);
    goto error;
}
zfs_uninit_libshare(hdl);

UTF8ONLY:
_utf = (int)intval;

NORMALIZE:
_normal = (int)intval;

to existing volumes, we have some additional
force.

S_TYPE_VOLUME && zhp != NULL) {
    t volsize = zfs_prop_get_int(zhp,
        S_PROP_VOLSIZE);
    t blocksize = zfs_prop_get_int(zhp,
        S_PROP_VOLEBLOCKSIZE);
    copies = zfs_prop_get_int(zhp, ZFS_PROP_COPIES);
    if [64];
    (prop) {
        S_PROP_RESERVATION:
        S_PROP_REFRESERVATION:
        if (intval >
            zvol_volsize_to_reservation_impl(volsize,
                blocksize, ncopies)) {
            if (intval > volsize) {
                zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
                    "'%s' is greater than current "
                    "volume size"), propname);
                (void) zfs_error(hdl, EZFS_BADPROP,

```

```

1225             errbuf);
1226         goto error;
1227     }
1228     break;
1229
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                                         proname, buf);
1238             (void) zfs_error(hdl, EZFS_BADPROP,
1239                             errbuf);
1240             goto error;
1241         }
1242
1243         if (intval == 0) {
1244             zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
1245                                         "'%s' cannot be zero"),
1246                                         proname);
1247             (void) zfs_error(hdl, EZFS_BADPROP,
1248                             errbuf);
1249             goto error;
1250         }
1251     }
1252     break;
1253 }
1254
1255 /*
1256 * If normalization was chosen, but no UTF8 choice was made,
1257 * enforce rejection of non-UTF8 names.
1258 */
1259
1260 /*
1261 * If normalization was chosen, but rejecting non-UTF8 names
1262 * was explicitly not chosen, it is an error.
1263 */
1264 if (chosen_normal > 0 && chosen_utf < 0) {
1265     if (nvlist_add_uint64(ret,
1266                           zfs_prop_to_name(ZFS_PROP_UTF8ONLY), 1) != 0) {
1267         (void) no_memory(hdl);
1268         goto error;
1269     }
1270 } else if (chosen_normal > 0 && chosen_utf == 0) {
1271     zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
1272                                 "'%s' must be set 'on' if normalization chosen"),
1273                                 zfs_prop_to_name(ZFS_PROP_UTF8ONLY));
1274     (void) zfs_error(hdl, EZFS_BADPROP, errbuf);
1275 }
1276
1277 return (ret);
1278
1279 error:
1280     nvlist_free(ret);
1281     return (NULL);
1282 }
1283
1284 int
1285 zfs_add_synthetic_resv(zfs_handle_t *zhp, nvlist_t *nvl)
1286 {
1287     uint64_t old_volsize;
1288     uint64_t new_volsize;
1289     uint64_t old_reservation;
1290     uint64_t new_reservation;
1291     zfs_prop_t resv_prop;

```

```

1291     uint64_t volblocksize;
1292     int ncopies;
1293     nvlist_t *props;
1294
1295     /*
1296      * If this is an existing volume, and someone is setting the volsize,
1297      * make sure that it matches the reservation, or add it if necessary.
1298      */
1299     old_volsize = zfs_prop_get_int(zhp, ZFS_PROP_VOLSIZE);
1300     if (zfs_which_resv_prop(zhp, &resv_prop) < 0)
1301         return (-1);
1302     old_reservation = zfs_prop_get_int(zhp, resv_prop);
1303     volblocksize = zfs_prop_get_int(zhp, ZFS_PROP_VOLBLOCKSIZE);
1304     ncopies = zfs_prop_get_int(zhp, ZFS_PROP_COPIES);
1305
1306     if ((zvol_volsize_to_reservation_impl(old_volsize, volblocksize,
1307                                           ncopies) != old_reservation) || nvlist_exists(nvl,
1308                                                 zfs_prop_to_name(resv_prop)))
1309     props = fnvlist_alloc();
1310     fnvlist_add_uint64(props, zfs_prop_to_name(ZFS_PROP_VOLBLOCKSIZE),
1311                         zfs_prop_get_int(zhp, ZFS_PROP_VOLBLOCKSIZE));
1312
1313     if ((zvol_volsize_to_reservation(old_volsize, props) !=
1314          old_reservation) || nvlist_exists(nvl,
1315                                           zfs_prop_to_name(resv_prop))) {
1316         fnvlist_free(props);
1317         return (0);
1318     }
1319     if (nvlist_lookup_uint64(nvl, zfs_prop_to_name(ZFS_PROP_VOLSIZE),
1320                             &new_volsize) != 0)
1321     &new_volsize) != 0) {
1322         fnvlist_free(props);
1323         return (-1);
1324     new_reservation = zvol_volsize_to_reservation_impl(new_volsize,
1325                                                       volblocksize, ncopies);
1326 }
1327 new_reservation = zvol_volsize_to_reservation(new_volsize, props);
1328 fnvlist_free(props);
1329
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
1337 unchanged_portion_omitted
1338
1339 */
1340 * Convert the zvol's volume size to an appropriate reservation. This is a
1341 * convenience front-end to zvol_volsize_to_reservation_impl.
1342 * Convert the zvol's volume size to an appropriate reservation.
1343 * Note: If this routine is updated, it is necessary to update the ZFS test
1344 * suite's shell version in reservation.kshlib.
1345 */
1346 uint64_t
1347 zvol_volsize_to_reservation(uint64_t volsize, nvlist_t *props)
1348 {
1349     uint64_t volblocksize;
1350     uint64_t numdb;
1351     uint64_t nblocks, volblocksize;
1352     int ncopies;
1353     char *strval;
1354
1355     if (nvlist_lookup_string(props,
1356                             zfs_prop_to_name(ZFS_PROP_COPIES), &strval) == 0)

```

```
4475         ncopies = atoi(strval);
4476     else
4477         ncopies = 1;
4478     if (nvlist_lookup_uint64(props,
4479         zfs_prop_to_name(ZFS_PROP_VOLBLOCKSIZE),
4480         &volblocksize) != 0)
4481         volblocksize = ZVOL_DEFAULT_BLOCKSIZE;
4482
4483     return (zvol_volsize_to_reservation_impl(volsize, volblocksize,
4484         ncopies));
4485 }
4486
4487 /*
4488 * Computes the required reservation to completely contain all blocks of a
4489 * zvol at a given volsize.
4490 */
4491 uint64_t
4492 zvol_volsize_to_reservation_impl(uint64_t volsize, uint64_t volblocksize,
4493     int ncopies)
4494 {
4495     uint64_t numdb;
4496     uint64_t nblocks;
4497
4498     nblocks = volsize/volblocksize;
4499     /* start with metadnode L0-L6 */
4500     numdb = 7;
4501     /* calculate number of indirects */
4502     while (nblocks > 1) {
4503         nblocks += DNODES_PER_LEVEL - 1;
4504         nblocks /= DNODES_PER_LEVEL;
4505         numdb += nblocks;
4506     }
4507     numdb *= MIN(SPA_DVAS_PER_BP, ncopies + 1);
4508     volsize *= ncopies;
4509     /*
4510      * this is exactly DN_MAX_INDBLKSHIFT when metadata isn't
4511      * compressed, but in practice they compress down to about
4512      * 1100 bytes
4513      */
4514     numdb *= 1ULL << DN_MAX_INDBLKSHIFT;
4515     volsize += numdb;
4516     return (volsize);
4517 }
```

unchanged_portion_omitted

new/usr/src/lib/libzfs/common/mapfile-vers

```
*****
5439 Tue Oct 15 13:59:52 2013
new/usr/src/lib/libzfs/common/mapfile-vers
4012 Upper limit of zfs set bounds check for refreservation 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 #
21 # Copyright (c) 2006, 2010, Oracle and/or its affiliates. All rights reserved.
22 # Copyright 2011 Nexenta Systems, Inc. All rights reserved.
23 # Copyright (c) 2012 by Delphix. All rights reserved.
24 #
25 # MAPFILE HEADER START
26 #
27 # WARNING: STOP NOW. DO NOT MODIFY THIS FILE.
28 # Object versioning must comply with the rules detailed in
29 #
30 #     usr/src/lib/README.mapfiles
31 #
32 # You should not be making modifications here until you've read the most current
33 # copy of that file. If you need help, contact a gatekeeper for guidance.
34 #
35 # MAPFILE HEADER END
36 #

38 $mapfile_version 2

40 SYMBOL_VERSION SUNWprivate_1.1 {
41     global:
42         fletcher_2_native;
43         fletcher_2_byteswap;
44         fletcher_4_native;
45         fletcher_4_byteswap;
46         fletcher_4_incremental_native;
47         fletcher_4_incremental_byteswap;
48         libzfs_add_handle;
49         libzfs_dataset_cmp;
50         libzfs_errno;
51         libzfs_error_action;
52         libzfs_error_description;
53         libzfs_fini;
54         libzfs_fru_compare;
55         libzfs_fru_devpath;
56         libzfs_fru_lookup;
57         libzfs_fru_notself;
58         libzfs_fru_refresh;
59         libzfs_init;
60         libzfs_mnttab_cache;
61         libzfs_print_on_error;
```

1

new/usr/src/lib/libzfs/common/mapfile-vers

```
62     spa_feature_table;
63     zfs_allocatable_devs;
64     zfs_asprintf;
65     zfs_clone;
66     zfs_close;
67     zfs_create;
68     zfs_create_ancestors;
69     zfs_dataset_exists;
70     zfs_deleg_share_nfs;
71     zfs_destroy;
72     zfs_destroy_snaps;
73     zfs_destroy_snaps_nv1;
74     zfs_expand_proplist;
75     zfs_get_handle;
76     zfs_get_holds;
77     zfs_get_name;
78     zfs_get_pool_handle;
79     zfs_get_user_props;
80     zfs_get_type;
81     zfs_handle_dup;
82     zfs_history_event_names;
83     zfs_hold;
84     zfs_is_mounted;
85     zfs_is_shared;
86     zfs_is_shared_nfs;
87     zfs_is_shared_smb;
88     zfs_iter_children;
89     zfs_iter_dependents;
90     zfs_iter_filesystems;
91     zfs_iter_root;
92     zfs_iter_snapshots;
93     zfs_iter_snapshots_sorted;
94     zfs_iter_snapspec;
95     zfs_mount;
96     zfs_name_to_prop;
97     zfs_name_valid;
98     zfs_nicenum;
99     zfs_nicestrtonum;
100    zfs_open;
101    zfs_path_to_zhandle;
102    zfs_promote;
103    zfs_prop_align_right;
104    zfs_prop_column_name;
105    zfs_prop_default_numeric;
106    zfs_prop_default_string;
107    zfs_prop_get;
108    zfs_prop_get_int;
109    zfs_prop_get_numeric;
110    zfs_prop_get_recvfd;
111    zfs_prop_get_table;
112    zfs_prop_get_userquota_int;
113    zfs_prop_get_userquota;
114    zfs_prop_get_written_int;
115    zfs_prop_get_written;
116    zfs_prop_inherit;
117    zfs_prop_inheritable;
118    zfs_prop_init;
119    zfs_prop_is_string;
120    zfs_prop_readonly;
121    zfs_prop_set;
122    zfs_prop_string_to_index;
123    zfs_prop_to_name;
124    zfs_prop_user;
125    zfs_prop_userquota;
126    zfs_prop_valid_for_type;
127    zfs_prop_values;
```

2

```

128     zfs_prop_written;
129     zfs_prune_proplist;
130     zfs_receive;
131     zfs_refresh_properties;
132     zfs_release;
133     zfs_rename;
134     zfs_rollback;
135     zfs_save_arguments;
136     zfs_send;
137     zfs_share;
138     zfs_shareall;
139     zfs_share_nfs;
140     zfs_share_smb;
141     zfs_show_diffs;
142     zfs_smb_acl_add;
143     zfs_smb_acl_purge;
144     zfs_smb_acl_remove;
145     zfs_smb_acl_rename;
146     zfs_snapshot;
147     zfs_snapshot_nv1;
148     zfs_spa_version;
149     zfs_spa_version_map;
150     zfs_type_to_name;
151     zfs_unmount;
152     zfs_umountall;
153     zfs_unshare;
154     zfs_unshare_nfs;
155     zfs_unshare_smb;
156     zfs_unshareall;
157     zfs_unshareall_bypath;
158     zfs_unshareall_nfs;
159     zfs_unshareall_smb;
160     zfs_userspace;
161     zfs_valid_proplist;
162     zfs_get_fsacl;
163     zfs_set_fsacl;
164     zfs_userquota_prop_prefixes;
165     zfs_zpl_version_map;
166     zpool_add;
167     zpool_clear;
168     zpool_clear_label;
169     zpool_close;
170     zpool_create;
171     zpool_destroy;
172     zpool_disable_datasets;
173     zpool_dump_ddt;
174     zpool_enable_datasets;
175     zpool_expand_proplist;
176     zpool_explain_recover;
177     zpool_export;
178     zpool_export_force;
179     zpool_find_import;
180     zpool_find_import_cached;
181     zpool_find_vdev;
182     zpool_find_vdev_by_physpath;
183     zpool_fru_set;
184     zpool_get_config;
185     zpool_get_errlog;
186     zpool_get_features;
187     zpool_get_handle;
188     zpool_get_history;
189     zpool_get_name;
190     zpool_get_physpath;
191     zpool_get_prop;
192     zpool_get_prop_int;
193     zpool_get_state;

```

```

194     zpool_get_status;
195     zpool_history_unpack;
196     zpool_import;
197     zpool_import_props;
198     zpool_import_status;
199     zpool_in_use;
200     zpool_is_bootable;
201     zpool_iter;
202     zpool_label_disk;
203     zpool_log_history;
204     zpool_mount_datasets;
205     zpool_name_to_prop;
206     zpool_obj_to_path;
207     zpool_open;
208     zpool_open_canfail;
209     zpool_print_unsup_feat;
210     zpool_prop_align_right;
211     zpool_prop_column_name;
212     zpool_prop_feature;
213     zpool_prop_get_feature;
214     zpool_prop_readonly;
215     zpool_prop_to_name;
216     zpool_prop_unsupported;
217     zpool_prop_values;
218     zpool_read_label;
219     zpool_refresh_stats;
220     zpool_reguid;
221     zpool_reopen;
222     zpool_scan;
223     zpool_search_import;
224     zpool_set_prop;
225     zpool_state_to_name;
226     zpool_umount_datasets;
227     zpool_upgrade;
228     zpool_vdev_attach;
229     zpool_vdev_clear;
230     zpool_vdev_degrade;
231     zpool_vdev_detach;
232     zpool_vdev_fault;
233     zpool_vdev_name;
234     zpool_vdev_offline;
235     zpool_vdev_online;
236     zpool_vdev_remove;
237     zpool_vdev_split;
238     zprop_free_list;
239     zprop_get_list;
240     zprop_iter;
241     zprop_print_one_property;
242     zprop_width;
243     zvol_check_dump_config;
244     zvol_volsize_to_reservation;
245     zvol_volsize_to_reservation_impl;
246     local:
247     *;
248 };

```

unchanged portion omitted