```
**********************************************************
   35753 Mon Jun  8 20:19:22 2015
new/usr/src/cmd/beadm/beadm.c
5679 be_sort_list(): Possible null pointer dereference
**********************************************************
```

```
   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) 2009, 2010, Oracle and/or its affiliates. All rights reserved.
  24  */

  26 /*
  27  * Copyright 2013 Nexenta Systems, Inc. All rights reserved.
  28  * Copyright 2015 Toomas Soome <tsoome@me.com>
  29  * Copyright 2015 Gary Mills
  30  */

  32 /*
  33  * System includes
  34  */

  36 #include <assert.h>
  37 #include <stdio.h>
  38 #include <strings.h>
  39 #include <libzfs.h>
  40 #include <locale.h>
  41 #include <langinfo.h>
  42 #include <stdlib.h>
  43 #include <wchar.h>
  44 #include <sys/types.h>

  46 #include "libbe.h"

  48 #ifndef lint
  49 #define _(x) gettext(x)
  50 #else
  51 #define _(x) (x)
  52 #endif

  54 #ifndef TEXT_DOMAIN
  55 #define TEXT_DOMAIN "SYS_TEST"
  56 #endif

  58 #define DT_BUF_LEN (128)
  59 #define NUM_COLS (6)

  61 static int be_do_activate(int argc, char **argv);
```



```
  62 static int be_do_create(int argc, char **argv);
  63 static int be_do_destroy(int argc, char **argv);
  64 static int be_do_list(int argc, char **argv);
  65 static int be_do_mount(int argc, char **argv);
  66 static int be_do_unmount(int argc, char **argv);
  67 static int be_do_rename(int argc, char **argv);
  68 static int be_do_rollback(int argc, char **argv);
  69 static void usage(void);

  71 /*
  72  * single column name/width output format description
  73  */
  74 struct col_info {
  75         const char *col_name;
  76         size_t width;
  77 };
_____unchanged_portion_omitted_

1080 static int
1081 be_do_list(int argc, char **argv)
1082 {
1083         be_node_list_t  *be_nodes = NULL;
1084         boolean_t       all = B_FALSE;
1085         boolean_t       dsets = B_FALSE;
1086         boolean_t       snaps = B_FALSE;
1087         boolean_t       parsable = B_FALSE;
1088         int             err = 1;
1089         int             c = 0;
1090         char            *be_name = NULL;
1091         be_sort_t       order = BE_SORT_UNSPECIFIED;

1093         while ((c = getopt(argc, argv, "adk:svHK:")) != -1) {
1094                 switch (c) {
1095                 case 'a':
1096                         all = B_TRUE;
1097                         break;
1098                 case 'd':
1099                         dsets = B_TRUE;
1100                         break;
1101                 case 'k':
1102                 case 'K':
1103                         if (order != BE_SORT_UNSPECIFIED) {
1104                                 (void) fprintf(stderr, _("Sort key can be "
1105                                     "specified only once.\n"));
1106                                 usage();
1107                                 return (1);
1108                         }
1109                         if (strcmp(optarg, "date") == 0) {
1110                                 if (c == 'k')
1111                                         order = BE_SORT_DATE;
1112                                 else
1113                                         order = BE_SORT_DATE_REV;
1114                                 break;
1115                         }
1116                         if (strcmp(optarg, "name") == 0) {
1117                                 if (c == 'k')
1118                                         order = BE_SORT_NAME;
1119                                 else
1120                                         order = BE_SORT_NAME_REV;
1121                                 break;
1122                         }
1123                         if (strcmp(optarg, "space") == 0) {
1124                                 if (c == 'k')
1125                                         order = BE_SORT_SPACE;
1126                                 else
1127                                         order = BE_SORT_SPACE_REV;
```

```
1128                                    break;
1129                            }
1130                            (void) fprintf(stderr, _("Unknown sort key: %s\n"),
1131                                optarg);
1132                            usage();
1133                            return (1);
1134                    case 's':
1135                            snaps = B_TRUE;
1136                            break;
1137                    case 'v':
1138                            libbe_print_errors(B_TRUE);
1139                            break;
1140                    case 'H':
1141                            parsable = B_TRUE;
1142                            break;
1143                    default:
1144                            usage();
1145                            return (1);
1146                    }
1147            }

1149            if (all) {
1150                    if (dsets) {
1151                            (void) fprintf(stderr, _("Invalid options: -a and %s "
1152                                "are mutually exclusive.\n"), "-d");
1153                            usage();
1154                            return (1);
1155                    }
1156                    if (snaps) {
1157                            (void) fprintf(stderr, _("Invalid options: -a and %s "
1158                                "are mutually exclusive.\n"), "-s");
1159                            usage();
1160                            return (1);
1161                    }

1163                    dsets = B_TRUE;
1164                    snaps = B_TRUE;
1165            }

1167            argc -= optind;
1168            argv += optind;


1171            if (argc == 1)
1172                    be_name = argv[0];

1174            err = be_list(be_name, &be_nodes);

1176            switch (err) {
1177            case BE_SUCCESS:
1178                    /* the default sort is ascending date, no need to sort twice */
1179                    if (order == BE_SORT_UNSPECIFIED)
1180                            order = BE_SORT_DATE;

1182                    if (order != BE_SORT_DATE) {
1183                            err = be_sort(&be_nodes, order);
1184                            if (err != BE_SUCCESS) {
1185                                    (void) fprintf(stderr, _("Unable to sort Boot "
1186                                        "Environment\n"));
1187                                    (void) fprintf(stderr, "%s\n",
1188                                        be_err_to_str(err));
1189                                    break;
1190                            }
1191                    }
1181                    if (order != BE_SORT_DATE)
1182                            be_sort(&be_nodes, order);
```



```
1193                    print_nodes(be_name, dsets, snaps, parsable, be_nodes);
1194                    break;
1195            case BE_ERR_BE_NOENT:
1196                    if (be_name == NULL)
1197                            (void) fprintf(stderr, _("No boot environments found "
1198                                "on this system.\n"));
1199                    else {
1200                            (void) fprintf(stderr, _("%s does not exist or appear "
1201                                "to be a valid BE.\nPlease check that the name of "
1202                                "the BE provided is correct.\n"), be_name);
1203                    }
1204                    break;
1205            default:
1206                    (void) fprintf(stderr, _("Unable to display Boot "
1207                        "Environment\n"));
1208                    (void) fprintf(stderr, "%s\n", be_err_to_str(err));
1209            }

1211            if (be_nodes != NULL)
1212                    be_free_list(be_nodes);
1213            return (err);
1214 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   37669 Mon Jun  8 20:19:23 2015
new/usr/src/lib/libbe/common/be_list.c
5679 be_sort_list(): Possible null pointer dereference
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright (c) 2008, 2010, Oracle and/or its affiliates. All rights reserved.
 */

/*
 * Copyright 2013 Nexenta Systems, Inc. All rights reserved.
 * Copyright 2015 Toomas Soome <tsoome@me.com>
 * Copyright 2015 Gary Mills
 */

#include <assert.h>
#include <libintl.h>
#include <libnvpair.h>
#include <libzfs.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>

#include <libbe.h>
#include <libbe_priv.h>

/*
 * Callback data used for zfs_iter calls.
 */
typedef struct list_callback_data {
        char *zpool_name;
        char *be_name;
        be_node_list_t *be_nodes_head;
        be_node_list_t *be_nodes;
        char current_be[MAXPATHLEN];
} list_callback_data_t;

/*
 * Private function prototypes
 */
```



```
static int be_add_children_callback(zfs_handle_t *zhp, void *data);
static int be_get_list_callback(zpool_handle_t *, void *);
static int be_get_node_data(zfs_handle_t *, be_node_list_t *, char *,
    const char *, char *, char *);
static int be_get_zone_node_data(be_node_list_t *, char *);
static int be_get_ds_data(zfs_handle_t *, char *, be_dataset_list_t *,
    be_node_list_t *);
static int be_get_ss_data(zfs_handle_t *, char *, be_snapshot_list_t *,
    be_node_list_t *);
static int be_sort_list(be_node_list_t **,
static void be_sort_list(be_node_list_t **,
    int (*)(const void *, const void *));
static int be_qsort_compare_BEs_name(const void *, const void *);
static int be_qsort_compare_BEs_name_rev(const void *, const void *);
static int be_qsort_compare_BEs_date(const void *, const void *);
static int be_qsort_compare_BEs_date_rev(const void *, const void *);
static int be_qsort_compare_BEs_space(const void *, const void *);
static int be_qsort_compare_BEs_space_rev(const void *, const void *);
static int be_qsort_compare_snapshots(const void *x, const void *y);
static int be_qsort_compare_datasets(const void *x, const void *y);
static void *be_list_alloc(int *, size_t);

/*
 * Private data.
 */
static char be_container_ds[MAXPATHLEN];
static boolean_t zone_be = B_FALSE;

/* ********************************************************************* */
/*                      Public Functions                                 */
/* ********************************************************************* */

/*
 * Function:    be_list
 * Description: Calls _be_list which finds all the BEs on the system and
 *              returns the datasets and snapshots belonging to each BE.
 *              Also data, such as dataset and snapshot properties,
 *              for each BE and their snapshots and datasets is
 *              returned. The data returned is as described in the
 *              be_dataset_list_t, be_snapshot_list_t and be_node_list_t
 *              structures.
 * Parameters:
 *              be_name - The name of the BE to look up.
 *                        If NULL a list of all BEs will be returned.
 *              be_nodes - A reference pointer to the list of BEs. The list
 *                         structure will be allocated by _be_list and must
 *                         be freed by a call to be_free_list. If there are no
 *                         BEs found on the system this reference will be
 *                         set to NULL.
 * Return:
 *              BE_SUCCESS - Success
 *              be_errno_t - Failure
 * Scope:
 *              Public
 */
int
be_list(char *be_name, be_node_list_t **be_nodes)
{
        int     ret = BE_SUCCESS;

        /* Initialize libzfs handle */
        if (!be_zfs_init())
                return (BE_ERR_INIT);

        /* Validate be_name if its not NULL */
        if (be_name != NULL) {
```

```
127                 if (!be_valid_be_name(be_name)) {
128                         be_print_err(gettext("be_list: "
129                             "invalid BE name %s\n"), be_name);
130                         return (BE_ERR_INVAL);
131                 }
132         }

134         ret = _be_list(be_name, be_nodes);

136         be_zfs_fini();

138         return (ret);
139 }

141 /*
142  * Function:    be_sort
143  * Description: Sort BE node list
144  * Parameters:
145  *              pointer to address of list head
146  *              sort order type
147  * Return:
148  *              BE_SUCCESS - Success
149  *              be_errno_t - Failure
146  * Returns:
147  *              nothing
150  * Side effect:
151  *              node list sorted by name
152  * Scope:
153  *              Public
154  */
155 int
153 void
156 be_sort(be_node_list_t **be_nodes, int order)
157 {
158         int (*compar)(const void *, const void *) = be_qsort_compare_BEs_date;

160         if (be_nodes == NULL)
161                 return (BE_ERR_INVAL);
159                 return;

163         switch (order) {
164         case BE_SORT_UNSPECIFIED:
165         case BE_SORT_DATE:
166                 compar = be_qsort_compare_BEs_date;
167                 break;
168         case BE_SORT_DATE_REV:
169                 compar = be_qsort_compare_BEs_date_rev;
170                 break;
171         case BE_SORT_NAME:
172                 compar = be_qsort_compare_BEs_name;
173                 break;
174         case BE_SORT_NAME_REV:
175                 compar = be_qsort_compare_BEs_name_rev;
176                 break;
177         case BE_SORT_SPACE:
178                 compar = be_qsort_compare_BEs_space;
179                 break;
180         case BE_SORT_SPACE_REV:
181                 compar = be_qsort_compare_BEs_space_rev;
182                 break;
183         default:
184                 be_print_err(gettext("be_sort: invalid sort order %d\n"),
185                     order);
186                 return (BE_ERR_INVAL);
184                 return;
187         }
```



```
189         return (be_sort_list(be_nodes, compar));
187         be_sort_list(be_nodes, compar);
190 }

192 /* ********************************************************************* */
193 /*                      Semi-Private Functions                           */
194 /* ********************************************************************* */

196 /*
197  * Function:    _be_list
198  * Description: This does the actual work described in be_list.
199  * Parameters:
200  *              be_name - The name of the BE to look up.
201  *                        If NULL a list of all BEs will be returned.
202  *              be_nodes - A reference pointer to the list of BEs. The list
203  *                         structure will be allocated here and must
204  *                         be freed by a call to be_free_list. If there are no
205  *                         BEs found on the system this reference will be
206  *                         set to NULL.
207  * Return:
208  *              BE_SUCCESS - Success
209  *              be_errno_t - Failure
210  * Scope:
211  *              Semi-private (library wide use only)
212  */
213 int
214 _be_list(char *be_name, be_node_list_t **be_nodes)
215 {
216         list_callback_data_t cb = { 0 };
217         be_transaction_data_t bt = { 0 };
218         int ret = BE_SUCCESS;
219         int sret;
220         zpool_handle_t *zphp;
221         char *rpool = NULL;
222         struct be_defaults be_defaults;

224         if (be_nodes == NULL)
225                 return (BE_ERR_INVAL);

227         be_get_defaults(&be_defaults);

229         if (be_find_current_be(&bt) != BE_SUCCESS) {
230                 /*
231                  * We were unable to find a currently booted BE which
232                  * probably means that we're not booted in a BE envoronment.
233                  * None of the BE's will be marked as the active BE.
234                  */
235                 (void) strcpy(cb.current_be, "-");
236         } else {
237                 (void) strncpy(cb.current_be, bt.obe_name,
238                     sizeof (cb.current_be));
239                 rpool = bt.obe_zpool;
240         }

242         /*
243          * If be_name is NULL we'll look for all BE's on the system.
244          * If not then we will only return data for the specified BE.
245          */
246         if (be_name != NULL)
247                 cb.be_name = strdup(be_name);

249         if (be_defaults.be_deflt_rpool_container && rpool != NULL) {
250                 if ((zphp = zpool_open(g_zfs, rpool)) == NULL) {
251                         be_print_err(gettext("be_list: failed to "
252                             "open rpool (%s): %s\n"), rpool,
```

```
253                             libzfs_error_description(g_zfs));
254                     free(cb.be_name);
255                     return (zfs_err_to_be_err(g_zfs));
256             }

258             ret = be_get_list_callback(zphp, &cb);
259     } else {
260             if ((zpool_iter(g_zfs, be_get_list_callback, &cb)) != 0) {
261                     if (cb.be_nodes_head != NULL) {
262                             be_free_list(cb.be_nodes_head);
263                             cb.be_nodes_head = NULL;
264                             cb.be_nodes = NULL;
265                     }
266                     ret = BE_ERR_BE_NOENT;
267             }
268     }

270     if (cb.be_nodes_head == NULL) {
271             if (be_name != NULL)
272                     be_print_err(gettext("be_list: BE (%s) does not "
273                         "exist\n"), be_name);
274             else
275                     be_print_err(gettext("be_list: No BE's found\n"));
276             ret = BE_ERR_BE_NOENT;
277     }

279     *be_nodes = cb.be_nodes_head;

281     free(cb.be_name);

283     sret = be_sort(be_nodes, BE_SORT_DATE);
280     be_sort(be_nodes, BE_SORT_DATE);

285     return ((ret == BE_SUCCESS) ? sret : ret);
282     return (ret);
286 }
```

*unchanged_portion_omitted_*

```
684 /*
685  * Function:    be_sort_list
686  * Description: Sort BE node list
687  * Parameters:
688  *              pointer to address of list head
689  *              compare function
690  * Return:
691  *              BE_SUCCESS - Success
692  *              be_errno_t - Failure
687  * Returns:
688  *              nothing
693  * Side effect:
694  *              node list sorted by name
695  * Scope:
696  *              Private
697  */
698 static int
694 static void
699 be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
700 {
701     int ret = BE_SUCCESS;
702     size_t ibe, nbe;
703     be_node_list_t *p = NULL;
704     be_node_list_t **ptrlist = NULL;
705     be_node_list_t **ptrtmp;

707     if (pstart == NULL) /* Nothing to sort */
708             return (BE_SUCCESS);
```



```
701     if (pstart == NULL)
702             return;
709     /* build array of linked list BE struct pointers */
710     for (p = *pstart, nbe = 0; p != NULL; nbe++, p = p->be_next_node) {
711             ptrtmp = realloc(ptrlist,
705             ptrlist = realloc(ptrlist,
712                 sizeof (be_node_list_t *) * (nbe + 2));
713             if (ptrtmp == NULL) { /* out of memory */
714                     be_print_err(gettext("be_sort_list: memory "
715                         "allocation failed\n"));
716                     ret = BE_ERR_NOMEM;
717                     goto free;
718             }
719             ptrlist = ptrtmp;
720             ptrlist[nbe] = p;
721     }
722     if (nbe == 0) /* Nothing to sort */
723             return (BE_SUCCESS);
709     if (nbe == 0)
710             return;
724     /* in-place list quicksort using qsort(3C) */
725     if (nbe > 1)    /* no sort if less than 2 BEs */
726             qsort(ptrlist, nbe, sizeof (be_node_list_t *), compar);

728     ptrlist[nbe] = NULL; /* add linked list terminator */
729     *pstart = ptrlist[0]; /* set new linked list header */
730     /* for each BE in list */
731     for (ibe = 0; ibe < nbe; ibe++) {
732             size_t k, ns;   /* subordinate index, count */

734             /* rewrite list pointer chain, including terminator */
735             ptrlist[ibe]->be_next_node = ptrlist[ibe + 1];
736             /* sort subordinate snapshots */
737             if (ptrlist[ibe]->be_node_num_snapshots > 1) {
738                     const size_t nmax = ptrlist[ibe]->be_node_num_snapshots;
739                     be_snapshot_list_t ** const slist =
740                         malloc(sizeof (be_snapshot_list_t *) * (nmax + 1));
741                     be_snapshot_list_t *p;

743                     if (slist == NULL) {
744                             ret = BE_ERR_NOMEM;
730                     if (slist == NULL)
745                             continue;
746                     }
747                     /* build array of linked list snapshot struct ptrs */
748                     for (ns = 0, p = ptrlist[ibe]->be_node_snapshots;
749                         ns < nmax && p != NULL;
750                         ns++, p = p->be_next_snapshot) {
751                             slist[ns] = p;
752                     }
753                     if (ns < 2)
754                             goto end_snapshot;
755                     slist[ns] = NULL; /* add terminator */
756                     /* in-place list quicksort using qsort(3C) */
757                     qsort(slist, ns, sizeof (be_snapshot_list_t *),
758                         be_qsort_compare_snapshots);
759                     /* rewrite list pointer chain, including terminator */
760                     ptrlist[ibe]->be_node_snapshots = slist[0];
761                     for (k = 0; k < ns; k++)
762                             slist[k]->be_next_snapshot = slist[k + 1];
763 end_snapshot:
764                     free(slist);
765             }
766             /* sort subordinate datasets */
767             if (ptrlist[ibe]->be_node_num_datasets > 1) {
768                     const size_t nmax = ptrlist[ibe]->be_node_num_datasets;
```

```
769                         be_dataset_list_t ** const slist =
770                             malloc(sizeof (be_dataset_list_t *) * (nmax + 1));
771                         be_dataset_list_t *p;

773                         if (slist == NULL) {
774                                 ret = BE_ERR_NOMEM;
758                         if (slist == NULL)
775                                 continue;
776                         }
777                         /* build array of linked list dataset struct ptrs */
778                         for (ns = 0, p = ptrlist[ibe]->be_node_datasets;
779                             ns < nmax && p != NULL;
780                             ns++, p = p->be_next_dataset) {
781                                 slist[ns] = p;
782                         }
783                         if (ns < 2) /* subordinate datasets < 2 - no sort */
784                                 goto end_dataset;
785                         slist[ns] = NULL; /* add terminator */
786                         /* in-place list quicksort using qsort(3C) */
787                         qsort(slist, ns, sizeof (be_dataset_list_t *),
788                             be_qsort_compare_datasets);
789                         /* rewrite list pointer chain, including terminator */
790                         ptrlist[ibe]->be_node_datasets = slist[0];
791                         for (k = 0; k < ns; k++)
792                                 slist[k]->be_next_dataset = slist[k + 1];
793 end_dataset:
794                         free(slist);
795                 }
796         }
797 free:
798         free(ptrlist);
799         return (ret);
800 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    8501 Mon Jun  8 20:19:23 2015
new/usr/src/lib/libbe/common/libbe.h
5679 be_sort_list(): Possible null pointer dereference
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright (c) 2008, 2010, Oracle and/or its affiliates. All rights reserved.
 */

/*
 * Copyright 2013 Nexenta Systems, Inc. All rights reserved.
 * Copyright 2015 Toomas Soome <tsoome@me.com>
 * Copyright 2015 Gary Mills
 */

#ifndef _LIBBE_H
#define _LIBBE_H

#include <libnvpair.h>
#include <uuid/uuid.h>
#include <libzfs.h>

#ifdef __cplusplus
extern "C" {
#endif

#define BE_ATTR_ORIG_BE_NAME    "orig_be_name"
#define BE_ATTR_ORIG_BE_POOL    "orig_be_pool"
#define BE_ATTR_SNAP_NAME       "snap_name"

#define BE_ATTR_NEW_BE_NAME     "new_be_name"
#define BE_ATTR_NEW_BE_POOL     "new_be_pool"
#define BE_ATTR_NEW_BE_DESC     "new_be_desc"
#define BE_ATTR_POLICY          "policy"
#define BE_ATTR_ZFS_PROPERTIES  "zfs_properties"

#define BE_ATTR_FS_NAMES        "fs_names"
#define BE_ATTR_FS_NUM          "fs_num"
#define BE_ATTR_SHARED_FS_NAMES "shared_fs_names"
#define BE_ATTR_SHARED_FS_NUM   "shared_fs_num"

#define BE_ATTR_MOUNTPOINT      "mountpoint"
#define BE_ATTR_MOUNT_FLAGS     "mount_flags"
#define BE_ATTR_UNMOUNT_FLAGS   "unmount_flags"
#define BE_ATTR_DESTROY_FLAGS   "destroy_flags"
```



```
#define BE_ATTR_ROOT_DS         "root_ds"
#define BE_ATTR_UUID_STR        "uuid_str"

#define BE_ATTR_ACTIVE          "active"
#define BE_ATTR_ACTIVE_ON_BOOT  "active_boot"
#define BE_ATTR_GLOBAL_ACTIVE   "global_active"
#define BE_ATTR_SPACE           "space_used"
#define BE_ATTR_DATASET         "dataset"
#define BE_ATTR_STATUS          "status"
#define BE_ATTR_DATE            "date"
#define BE_ATTR_MOUNTED         "mounted"

/*
 * libbe error codes
 *
 * NOTE: there is a copy of this enum in beadm/messages.py. To keep these
 *       in sync please make sure to add any new error messages at the end
 *       of this enumeration.
 */
enum {
        BE_SUCCESS = 0,
        BE_ERR_ACCESS = 4000,   /* permission denied */
        BE_ERR_ACTIVATE_CURR,   /* Activation of current BE failed */
        BE_ERR_AUTONAME,        /* auto naming failed */
        BE_ERR_BE_NOENT,        /* No such BE */
        BE_ERR_BUSY,            /* mount busy */
        BE_ERR_CANCELED,        /* operation canceled */
        BE_ERR_CLONE,           /* BE clone failed */
        BE_ERR_COPY,            /* BE copy failed */
        BE_ERR_CREATDS,         /* dataset creation failed */
        BE_ERR_CURR_BE_NOT_FOUND,       /* Can't find current BE */
        BE_ERR_DESTROY,         /* failed to destroy BE or snapshot */
        BE_ERR_DEMOTE,          /* BE demotion failed */
        BE_ERR_DSTYPE,          /* invalid dataset type */
        BE_ERR_BE_EXISTS,       /* BE exists */
        BE_ERR_INIT,            /* be_zfs_init failed */
        BE_ERR_INTR,            /* interupted system call */
        BE_ERR_INVAL,           /* invalid argument */
        BE_ERR_INVALPROP,       /* invalid property for dataset */
        BE_ERR_INVALMOUNTPOINT, /* Unexpected mountpoint */
        BE_ERR_MOUNT,           /* mount failed */
        BE_ERR_MOUNTED,         /* already mounted */
        BE_ERR_NAMETOOLONG,     /* name > BUFSIZ */
        BE_ERR_NOENT,           /* Doesn't exist */
        BE_ERR_POOL_NOENT,      /* No such pool */
        BE_ERR_NODEV,           /* No such device */
        BE_ERR_NOTMOUNTED,      /* File system not mounted */
        BE_ERR_NOMEM,           /* not enough memory */
        BE_ERR_NONINHERIT,      /* property is not inheritable for BE dataset */
        BE_ERR_NXIO,            /* No such device or address */
        BE_ERR_NOSPC,           /* No space on device */
        BE_ERR_NOTSUP,          /* Operation not supported */
        BE_ERR_OPEN,            /* open failed */
        BE_ERR_PERM,            /* Not owner */
        BE_ERR_UNAVAIL,         /* The BE is currently unavailable */
        BE_ERR_PROMOTE,         /* BE promotion failed */
        BE_ERR_ROFS,            /* read only file system */
        BE_ERR_READONLYDS,      /* read only dataset */
        BE_ERR_READONLYPROP,    /* read only property */
        BE_ERR_SS_EXISTS,       /* snapshot exists */
        BE_ERR_SS_NOENT,        /* No such snapshot */
        BE_ERR_UMOUNT,          /* unmount failed */
        BE_ERR_UMOUNT_CURR_BE,  /* Can't unmount current BE */
        BE_ERR_UMOUNT_SHARED,   /* unmount of shared File System failed */
        BE_ERR_UNKNOWN,         /* Unknown error */
        BE_ERR_ZFS,             /* ZFS returned an error */
```

```
128         BE_ERR_DESTROY_CURR_BE, /* Cannot destroy current BE */
129         BE_ERR_GEN_UUID,        /* Failed to generate uuid */
130         BE_ERR_PARSE_UUID,      /* Failed to parse uuid */
131         BE_ERR_NO_UUID,         /* BE has no uuid */
132         BE_ERR_ZONE_NO_PARENTBE,    /* Zone root dataset has no parent uuid */
133         BE_ERR_ZONE_MULTIPLE_ACTIVE, /* Zone has multiple active roots */
134         BE_ERR_ZONE_NO_ACTIVE_ROOT, /* Zone has no active root for this BE */
135         BE_ERR_ZONE_ROOT_NOT_LEGACY, /* Zone root dataset mntpt is not legacy */
136         BE_ERR_NO_MOUNTED_ZONE, /* Zone not mounted in alternate BE */
137         BE_ERR_MOUNT_ZONEROOT,  /* Failed to mount a zone root */
138         BE_ERR_UMOUNT_ZONEROOT, /* Failed to unmount a zone root */
139         BE_ERR_ZONES_UNMOUNT,   /* Unable to unmount a zone. */
140         BE_ERR_FAULT,           /* Bad Address */
141         BE_ERR_RENAME_ACTIVE,   /* Renaming the active BE is not supported */
142         BE_ERR_NO_MENU,         /* Missing boot menu file */
143         BE_ERR_DEV_BUSY,        /* Device is Busy */
144         BE_ERR_BAD_MENU_PATH,   /* Invalid path for menu.lst file */
145         BE_ERR_ZONE_SS_EXISTS,  /* zone snapshot already exists */
146         BE_ERR_ADD_SPLASH_ICT,  /* Add_splash_image ICT failed */
147         BE_ERR_BOOTFILE_INST,   /* Error installing boot files */
148         BE_ERR_EXTCMD           /* External command error */
149 } be_errno_t;
______unchanged_portion_omitted_

219 /*
220  * BE functions
221  */
222 int be_init(nvlist_t *);
223 int be_destroy(nvlist_t *);
224 int be_copy(nvlist_t *);

226 int be_mount(nvlist_t *);
227 int be_unmount(nvlist_t *);

229 int be_rename(nvlist_t *);

231 int be_activate(nvlist_t *);

233 int be_create_snapshot(nvlist_t *);
234 int be_destroy_snapshot(nvlist_t *);
235 int be_rollback(nvlist_t *);

237 /*
238  * Functions for listing and getting information about existing BEs.
239  */
240 int be_list(char *, be_node_list_t **);
241 void be_free_list(be_node_list_t *);
242 int be_max_avail(char *, uint64_t *);
243 char *be_err_to_str(int);
244 int be_sort(be_node_list_t **, int);
243 void be_sort(be_node_list_t **, int);

246 /*
247  * Library functions
248  */
249 void libbe_print_errors(boolean_t);

251 #ifdef __cplusplus
252 }
______unchanged_portion_omitted_
```