

```
*****
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 */
21 /*
22 * Copyright (c) 2009, 2010, Oracle and/or its affiliates. All rights reserved.
23 */
24 /*
25 * Copyright 2013 Nexenta Systems, Inc. All rights reserved.
26 * Copyright 2015 Toomas Soome <tsoome@me.com>
27 * Copyright 2015 Gary Mills
28 */
29 /*
30 */
31 /*
32 * System includes
33 */
34 */
35 #include <assert.h>
36 #include <stdio.h>
37 #include <strings.h>
38 #include <libzfs.h>
39 #include <locale.h>
40 #include <langinfo.h>
41 #include <stdlib.h>
42 #include <wchar.h>
43 #include <sys/types.h>
44
45 #include "libbe.h"
46
47 #ifndef lint
48 #define _(x) gettext(x)
49 #else
50 #define _(x) (x)
51 #endif
52
53 #ifndef TEXT_DOMAIN
54 #define TEXT_DOMAIN "SYS_TEST"
55 #endif
56
57 #define DT_BUF_LEN (128)
58 #define NUM_COLS (6)
59
60 static int be_do_activate(int argc, char **argv);
61
```

```
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 };
78 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;

1092     while ((c = getopt(argc, argv, "adk:svHK:")) != -1) {
1093         switch (c) {
1094             case 'a':
1095                 all = B_TRUE;
1096                 break;
1097             case 'd':
1098                 dsets = B_TRUE;
1099                 break;
1100             case 'k':
1101                 case 'K':
1102                     if (order != BE_SORT_UNSPECIFIED) {
1103                         (void) fprintf(stderr, _("Sort key can be "
1104                         "specified only once.\n"));
1105                         usage();
1106                         return (1);
1107                     }
1108                 if (strcmp(optarg, "date") == 0) {
1109                     if (c == 'k')
1110                         order = BE_SORT_DATE;
1111                     else
1112                         order = BE_SORT_DATE_REV;
1113                     break;
1114                 }
1115                 if (strcmp(optarg, "name") == 0) {
1116                     if (c == 'k')
1117                         order = BE_SORT_NAME;
1118                     else
1119                         order = BE_SORT_NAME_REV;
1120                     break;
1121                 }
1122                 if (strcmp(optarg, "space") == 0) {
1123                     if (c == 'k')
1124                         order = BE_SORT_SPACE;
1125                     else
1126                         order = BE_SORT_SPACE_REV;
1127             }
1128         }
1129     }
1130 }
```

```

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 }
1148 if (all) {
1149     if (dsets) {
1150         (void) fprintf(stderr, _("Invalid options: -a and %s "
1151                             "are mutually exclusive.\n"), "-d");
1152         usage();
1153         return (1);
1154     }
1155     if (snaps) {
1156         (void) fprintf(stderr, _("Invalid options: -a and %s "
1157                             "are mutually exclusive.\n"), "-s");
1158         usage();
1159         return (1);
1160     }
1161 }
1162 dsets = B_TRUE;
1163 snaps = B_TRUE;
1164 }
1165
1166 argc -= optind;
1167 argv += optind;
1168
1169 if (argc == 1)
1170     be_name = argv[0];
1171
1172 err = be_list(be_name, &be_nodes);
1173
1174 switch (err) {
1175 case BE_SUCCESS:
1176     /* the default sort is ascending date, no need to sort twice */
1177     if (order == BE_SORT_UNSPECIFIED)
1178         order = BE_SORT_DATE;
1179
1180     if (order != BE_SORT_DATE) {
1181         err = be_sort(&be_nodes, order);
1182         if (err != BE_SUCCESS) {
1183             (void) fprintf(stderr, _("Unable to sort Boot "
1184                             "Environment\n"));
1185             (void) fprintf(stderr, "%s\n",
1186                         be_err_to_str(err));
1187             break;
1188         }
1189     }
1190 }
1191 if (order != BE_SORT_DATE)
1192     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 }
1210 if (be_nodes != NULL)
1211     be_free_list(be_nodes);
1212 return (err);
1213
1214 }  

1215 unchanged_portion_omitted

```

new/usr/src/lib/libbe/common/be\_list.c

```
*****
3769 Mon Jun  8 20:19:23 2015
new/usr/src/lib/libbe/common/be_list.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) 2008, 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 #include <assert.h>
33 #include <libintl.h>
34 #include <libnvpair.h>
35 #include <libzfs.h>
36 #include <stdio.h>
37 #include <stdlib.h>
38 #include <string.h>
39 #include <strings.h>
40 #include <sys/types.h>
41 #include <sys/stat.h>
42 #include <unistd.h>
43 #include <errno.h>
45 #include <libbe.h>
46 #include <libbe_priv.h>
48 /*
49  * Callback data used for zfs_iter calls.
50 */
51 typedef struct list_callback_data {
52     char *zpool_name;
53     char *be_name;
54     be_node_list_t *be_nodes_head;
55     be_node_list_t *be_nodes;
56     char current_be[MAXPATHLEN];
57 } list_callback_data_t;
59 /*
60  * Private function prototypes
61 */
*****
```

1

new/usr/src/lib/libbe/common/be\_list.c

```
62 static int be_add_children_callback(zfs_handle_t *zhp, void *data);
63 static int be_get_list_callback(zpool_handle_t *, void *);
64 static int be_get_node_data(zfs_handle_t *, be_node_list_t *, char *,
65     const char *, char *, char *);
66 static int be_get_zone_node_data(be_node_list_t *, char *);
67 static int be_get_ds_data(zfs_handle_t *, char *, be_dataset_list_t *,
68     be_node_list_t *);
69 static int be_get_ss_data(zfs_handle_t *, char *, be_snapshot_list_t *,
70     be_node_list_t *);
71 static int be_sort_list(be_node_list_t **,
72     static void be_sort_list(be_node_list_t **,
73     int (*) (const void *, const void *));
74 static int be_qsort_compare_BEs_name(const void *, const void *);
75 static int be_qsort_compare_BEs_name_rev(const void *, const void *);
76 static int be_qsort_compare_BEs_date(const void *, const void *);
77 static int be_qsort_compare_BEs_date_rev(const void *, const void *);
78 static int be_qsort_compare_BEs_space(const void *, const void *);
79 static int be_qsort_compare_BEs_space_rev(const void *, const void *);
80 static int be_qsort_compare_snapshots(const void *x, const void *y);
81 static void *be_list_alloc(int *, size_t);
83 /*
84  * Private data.
85 */
86 static char be_container_ds[MAXPATHLEN];
87 static boolean_t zone_be = B_FALSE;
89 /* **** */
90 /* Public Functions */
91 /* **** */
93 /*
94  * Function: be_list
95  * Description: Calls _be_list which finds all the BEs on the system and
96  * returns the datasets and snapshots belonging to each BE.
97  * Also data, such as dataset and snapshot properties,
98  * for each BE and their snapshots and datasets is
99  * returned. The data returned is as described in the
100 * be_dataset_list_t, be_snapshot_list_t and be_node_list_t
101 * structures.
102 * Parameters:
103 *      be_name - The name of the BE to look up.
104 *              If NULL a list of all BEs will be returned.
105 *      be_nodes - A reference pointer to the list of BEs. The list
106 *              structure will be allocated by _be_list and must
107 *              be freed by a call to be_free_list. If there are no
108 *              BEs found on the system this reference will be
109 *              set to NULL.
110 *      Return:
111 *          BE_SUCCESS - Success
112 *          be_errno_t - Failure
113 *      Scope:
114 *          Public
115 */
116 int
117 be_list(char *be_name, be_node_list_t **be_nodes)
118 {
119     int ret = BE_SUCCESS;
120     /* Initialize libzfs handle */
121     if (!be_zfs_init())
122         return (BE_ERR_INIT);
123
124     /* Validate be_name if its not NULL */
125     if (be_name != NULL) {
```

2

```
new/usr/src/lib/libbe/common/be_list.c

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 * Returns:
148 *           BE_SUCCESS - Success
149 *           be_errno_t - Failure
150 * Side effect:
151 *           node list sorted by name
152 * Scope:
153 *           Public
154 */
155 int
156 void
157 be_sort(be_node_list_t **be_nodes, int order)
158 {
159     int (*compar)(const void *, const void *) = be_qsort_compare_BEs_date;
160
161     if (be_nodes == NULL)
162         return (BE_ERR_INVAL);
163     return;
164
165     switch (order) {
166     case BE_SORT_UNSPECIFIED:
167     case BE_SORT_DATE:
168         compar = be_qsort_compare_BEs_date;
169         break;
170     case BE_SORT_DATE_REV:
171         compar = be_qsort_compare_BEs_date_rev;
172         break;
173     case BE_SORT_NAME:
174         compar = be_qsort_compare_BEs_name;
175         break;
176     case BE_SORT_NAME_REV:
177         compar = be_qsort_compare_BEs_name_rev;
178         break;
179     case BE_SORT_SPACE:
180         compar = be_qsort_compare_BEs_space;
181         break;
182     case BE_SORT_SPACE_REV:
183         compar = be_qsort_compare_BEs_space_rev;
184         break;
185     default:
186         be_print_err(gettext("be_sort: invalid sort order %d\n"),
187                     order);
188         return (BE_ERR_INVAL);
189     }
190 }
```

```
3      new/usr/src/lib/libbe/common/be_list.c

189      return (be_sort_list(be_nodes, compar));
190  }
191  /* **** Semi-Private Functions **** */
192  /* **** */
193  /*
194  * Function: _be_list
195  * Description: This does the actual work described in be_list.
196  * Parameters:
197  *   be_name - The name of the BE to look up.
198  *           If NULL a list of all BEs will be returned.
199  *   be_nodes - A reference pointer to the list of BEs. The list
200  *           structure will be allocated here and must
201  *           be freed by a call to be_free_list. If there are no
202  *           BEs found on the system this reference will be
203  *           set to NULL.
204  *
205  * Return:
206  *   BE_SUCCESS - Success
207  *   be_errno_t - Failure
208  * Scope:
209  *   Semi-private (library wide use only)
210  */
211 int _be_list(char *be_name, be_node_list_t **be_nodes)
212 {
213     list_callback_data_t cb = { 0 };
214     be_transaction_data_t bt = { 0 };
215     int ret = BE_SUCCESS;
216     int sret;
217     zpool_handle_t *zphp;
218     char *rpool = NULL;
219     struct be_defaults be_defaults;
220
221     if (be_nodes == NULL)
222         return (BE_ERR_INVAL);
223
224     be_get_defaults(&be_defaults);
225
226     if (be_find_current_be(&bt) != BE_SUCCESS) {
227         /*
228          * We were unable to find a currently booted BE which
229          * probably means that we're not booted in a BE environment.
230          * None of the BE's will be marked as the active BE.
231          */
232         (void) strcpy(cb.current_be, "-");
233     } else {
234         (void) strncpy(cb.current_be, bt.obe_name,
235                        sizeof (cb.current_be));
236         rpool = bt.obe_zpool;
237     }
238
239     /*
240      * If be_name is NULL we'll look for all BE's on the system.
241      * If not then we will only return data for the specified BE.
242      */
243     if (be_name != NULL)
244         cb.be_name = strdup(be_name);
245
246     if (be_defaults.be_deflt_rpool_container && rpool != NULL) {
247         if ((zphp = zpool_open(g_zfs, rpool)) == NULL) {
248             be_print_err(gettext("be_list: failed to "
249                                "open rpool (%s): %s\n"), rpool);
250         }
251     }
252 }
```

```

253         libzfs_error_description(g_zfs));
254         free(cb.be_name);
255     }
256
257     if (ret == BE_SUCCESS)
258     } else {
259         if ((zpool_iter(g_zfs, be_get_list_callback, &cb)) != 0) {
260             if (cb.be_nodes_head != NULL) {
261                 be_free_list(cb.be_nodes_head);
262                 cb.be_nodes_head = NULL;
263                 cb.be_nodes = NULL;
264             }
265             ret = BE_ERR_BE_NOENT;
266         }
267     }
268
269     if (cb.be_nodes_head == NULL) {
270         if (be_name != NULL)
271             be_print_err(gettext("be_list: BE (%s) does not "
272                                 "exist\n"), be_name);
273         else
274             be_print_err(gettext("be_list: No BE's found\n"));
275         ret = BE_ERR_BE_NOENT;
276     }
277
278     *be_nodes = cb.be_nodes_head;
279
280     free(cb.be_name);
281
282     sret = be_sort(be_nodes, BE_SORT_DATE);
283     be_sort(be_nodes, BE_SORT_DATE);
284
285     return ((ret == BE_SUCCESS) ? sret : ret);
286 }
unchanged_portion_omitted_
287
288 static int
289 static void
290 be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
291 {
292     int ret = BE_SUCCESS;
293     size_t ibe, nbe;
294     be_node_list_t *p = NULL;
295     be_node_list_t **ptrlist = NULL;
296     be_node_list_t **ptrtmp;
297
298     if (pstart == NULL) /* Nothing to sort */
299         return (BE_SUCCESS);
300
301     /* Function: be_sort_list
302      * Description: Sort BE node list
303      * Parameters:
304      *   pointer to address of list head
305      *   compare function
306      *
307      * Return:
308      *   BE_SUCCESS - Success
309      *   be_errno_t - Failure
310      *
311      * Returns:
312      *   nothing
313      *
314      * Side effect:
315      *   node list sorted by name
316      *
317      * Scope:
318      *   Private
319      */
320
321     static int
322     static void
323     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
324     {
325         int ret = BE_SUCCESS;
326         size_t ibe, nbe;
327         be_node_list_t *p = NULL;
328         be_node_list_t **ptrlist = NULL;
329         be_node_list_t **ptrtmp;
330
331         if (pstart == NULL) /* Nothing to sort */
332             return (BE_SUCCESS);
333
334         /* Function: be_sort_list
335          * Description: Sort BE node list
336          * Parameters:
337          *   pointer to address of list head
338          *   compare function
339          *
340          * Return:
341          *   BE_SUCCESS - Success
342          *   be_errno_t - Failure
343          *
344          * Returns:
345          *   nothing
346          *
347          * Side effect:
348          *   node list sorted by name
349          *
350          * Scope:
351          *   Private
352          */
353
354     static int
355     static void
356     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
357     {
358         int ret = BE_SUCCESS;
359         size_t ibe, nbe;
360         be_node_list_t *p = NULL;
361         be_node_list_t **ptrlist = NULL;
362         be_node_list_t **ptrtmp;
363
364         if (pstart == NULL) /* Nothing to sort */
365             return (BE_SUCCESS);
366
367         /* Function: be_sort_list
368          * Description: Sort BE node list
369          * Parameters:
370          *   pointer to address of list head
371          *   compare function
372          *
373          * Return:
374          *   BE_SUCCESS - Success
375          *   be_errno_t - Failure
376          *
377          * Returns:
378          *   nothing
379          *
380          * Side effect:
381          *   node list sorted by name
382          *
383          * Scope:
384          *   Private
385          */
386
387     static int
388     static void
389     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
390     {
391         int ret = BE_SUCCESS;
392         size_t ibe, nbe;
393         be_node_list_t *p = NULL;
394         be_node_list_t **ptrlist = NULL;
395         be_node_list_t **ptrtmp;
396
397         if (pstart == NULL) /* Nothing to sort */
398             return (BE_SUCCESS);
399
400         /* Function: be_sort_list
401          * Description: Sort BE node list
402          * Parameters:
403          *   pointer to address of list head
404          *   compare function
405          *
406          * Return:
407          *   BE_SUCCESS - Success
408          *   be_errno_t - Failure
409          *
410          * Returns:
411          *   nothing
412          *
413          * Side effect:
414          *   node list sorted by name
415          *
416          * Scope:
417          *   Private
418          */
419
420     static int
421     static void
422     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
423     {
424         int ret = BE_SUCCESS;
425         size_t ibe, nbe;
426         be_node_list_t *p = NULL;
427         be_node_list_t **ptrlist = NULL;
428         be_node_list_t **ptrtmp;
429
430         if (pstart == NULL) /* Nothing to sort */
431             return (BE_SUCCESS);
432
433         /* Function: be_sort_list
434           * Description: Sort BE node list
435           * Parameters:
436           *   pointer to address of list head
437           *   compare function
438           *
439           * Return:
440           *   BE_SUCCESS - Success
441           *   be_errno_t - Failure
442           *
443           * Returns:
444           *   nothing
445           *
446           * Side effect:
447           *   node list sorted by name
448           *
449           * Scope:
450           *   Private
451           */
452
453     static int
454     static void
455     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
456     {
457         int ret = BE_SUCCESS;
458         size_t ibe, nbe;
459         be_node_list_t *p = NULL;
460         be_node_list_t **ptrlist = NULL;
461         be_node_list_t **ptrtmp;
462
463         if (pstart == NULL) /* Nothing to sort */
464             return (BE_SUCCESS);
465
466         /* Function: be_sort_list
467           * Description: Sort BE node list
468           * Parameters:
469           *   pointer to address of list head
470           *   compare function
471           *
472           * Return:
473           *   BE_SUCCESS - Success
474           *   be_errno_t - Failure
475           *
476           * Returns:
477           *   nothing
478           *
479           * Side effect:
480           *   node list sorted by name
481           *
482           * Scope:
483           *   Private
484           */
485
486     static int
487     static void
488     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
489     {
490         int ret = BE_SUCCESS;
491         size_t ibe, nbe;
492         be_node_list_t *p = NULL;
493         be_node_list_t **ptrlist = NULL;
494         be_node_list_t **ptrtmp;
495
496         if (pstart == NULL) /* Nothing to sort */
497             return (BE_SUCCESS);
498
499         /* Function: be_sort_list
500           * Description: Sort BE node list
501           * Parameters:
502           *   pointer to address of list head
503           *   compare function
504           *
505           * Return:
506           *   BE_SUCCESS - Success
507           *   be_errno_t - Failure
508           *
509           * Returns:
510           *   nothing
511           *
512           * Side effect:
513           *   node list sorted by name
514           *
515           * Scope:
516           *   Private
517           */
518
519     static int
520     static void
521     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
522     {
523         int ret = BE_SUCCESS;
524         size_t ibe, nbe;
525         be_node_list_t *p = NULL;
526         be_node_list_t **ptrlist = NULL;
527         be_node_list_t **ptrtmp;
528
529         if (pstart == NULL) /* Nothing to sort */
530             return (BE_SUCCESS);
531
532         /* Function: be_sort_list
533           * Description: Sort BE node list
534           * Parameters:
535           *   pointer to address of list head
536           *   compare function
537           *
538           * Return:
539           *   BE_SUCCESS - Success
540           *   be_errno_t - Failure
541           *
542           * Returns:
543           *   nothing
544           *
545           * Side effect:
546           *   node list sorted by name
547           *
548           * Scope:
549           *   Private
550           */
551
552     static int
553     static void
554     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
555     {
556         int ret = BE_SUCCESS;
557         size_t ibe, nbe;
558         be_node_list_t *p = NULL;
559         be_node_list_t **ptrlist = NULL;
560         be_node_list_t **ptrtmp;
561
562         if (pstart == NULL) /* Nothing to sort */
563             return (BE_SUCCESS);
564
565         /* Function: be_sort_list
566           * Description: Sort BE node list
567           * Parameters:
568           *   pointer to address of list head
569           *   compare function
570           *
571           * Return:
572           *   BE_SUCCESS - Success
573           *   be_errno_t - Failure
574           *
575           * Returns:
576           *   nothing
577           *
578           * Side effect:
579           *   node list sorted by name
580           *
581           * Scope:
582           *   Private
583           */
584
585     static int
586     static void
587     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
588     {
589         int ret = BE_SUCCESS;
590         size_t ibe, nbe;
591         be_node_list_t *p = NULL;
592         be_node_list_t **ptrlist = NULL;
593         be_node_list_t **ptrtmp;
594
595         if (pstart == NULL) /* Nothing to sort */
596             return (BE_SUCCESS);
597
598         /* Function: be_sort_list
599           * Description: Sort BE node list
600           * Parameters:
601           *   pointer to address of list head
602           *   compare function
603           *
604           * Return:
605           *   BE_SUCCESS - Success
606           *   be_errno_t - Failure
607           *
608           * Returns:
609           *   nothing
610           *
611           * Side effect:
612           *   node list sorted by name
613           *
614           * Scope:
615           *   Private
616           */
617
618     static int
619     static void
620     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
621     {
622         int ret = BE_SUCCESS;
623         size_t ibe, nbe;
624         be_node_list_t *p = NULL;
625         be_node_list_t **ptrlist = NULL;
626         be_node_list_t **ptrtmp;
627
628         if (pstart == NULL) /* Nothing to sort */
629             return (BE_SUCCESS);
630
631         /* Function: be_sort_list
632           * Description: Sort BE node list
633           * Parameters:
634           *   pointer to address of list head
635           *   compare function
636           *
637           * Return:
638           *   BE_SUCCESS - Success
639           *   be_errno_t - Failure
640           *
641           * Returns:
642           *   nothing
643           *
644           * Side effect:
645           *   node list sorted by name
646           *
647           * Scope:
648           *   Private
649           */
650
651     static int
652     static void
653     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
654     {
655         int ret = BE_SUCCESS;
656         size_t ibe, nbe;
657         be_node_list_t *p = NULL;
658         be_node_list_t **ptrlist = NULL;
659         be_node_list_t **ptrtmp;
660
661         if (pstart == NULL) /* Nothing to sort */
662             return (BE_SUCCESS);
663
664         /* Function: be_sort_list
665           * Description: Sort BE node list
666           * Parameters:
667           *   pointer to address of list head
668           *   compare function
669           *
670           * Return:
671           *   BE_SUCCESS - Success
672           *   be_errno_t - Failure
673           *
674           * Returns:
675           *   nothing
676           *
677           * Side effect:
678           *   node list sorted by name
679           *
680           * Scope:
681           *   Private
682           */
683
684     static int
685     static void
686     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
687     {
688         int ret = BE_SUCCESS;
689         size_t ibe, nbe;
690         be_node_list_t *p = NULL;
691         be_node_list_t **ptrlist = NULL;
692         be_node_list_t **ptrtmp;
693
694         if (pstart == NULL) /* Nothing to sort */
695             return (BE_SUCCESS);
696
697         /* Function: be_sort_list
698           * Description: Sort BE node list
699           * Parameters:
700           *   pointer to address of list head
701           *   compare function
702           *
703           * Return:
704           *   BE_SUCCESS - Success
705           *   be_errno_t - Failure
706           *
707           * Returns:
708           *   nothing
709           *
710           * Side effect:
711           *   node list sorted by name
712           *
713           * Scope:
714           *   Private
715           */
716
717     static int
718     static void
719     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
720     {
721         int ret = BE_SUCCESS;
722         size_t ibe, nbe;
723         be_node_list_t *p = NULL;
724         be_node_list_t **ptrlist = NULL;
725         be_node_list_t **ptrtmp;
726
727         if (pstart == NULL) /* Nothing to sort */
728             return (BE_SUCCESS);
729
730         /* Function: be_sort_list
731           * Description: Sort BE node list
732           * Parameters:
733           *   pointer to address of list head
734           *   compare function
735           *
736           * Return:
737           *   BE_SUCCESS - Success
738           *   be_errno_t - Failure
739           *
740           * Returns:
741           *   nothing
742           *
743           * Side effect:
744           *   node list sorted by name
745           *
746           * Scope:
747           *   Private
748           */
749
750     static int
751     static void
752     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
753     {
754         int ret = BE_SUCCESS;
755         size_t ibe, nbe;
756         be_node_list_t *p = NULL;
757         be_node_list_t **ptrlist = NULL;
758         be_node_list_t **ptrtmp;
759
760         if (pstart == NULL) /* Nothing to sort */
761             return (BE_SUCCESS);
762
763         /* Function: be_sort_list
764           * Description: Sort BE node list
765           * Parameters:
766           *   pointer to address of list head
767           *   compare function
768           *
769           * Return:
770           *   BE_SUCCESS - Success
771           *   be_errno_t - Failure
772           *
773           * Returns:
774           *   nothing
775           *
776           * Side effect:
777           *   node list sorted by name
778           *
779           * Scope:
780           *   Private
781           */
782
783     static int
784     static void
785     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
786     {
787         int ret = BE_SUCCESS;
788         size_t ibe, nbe;
789         be_node_list_t *p = NULL;
790         be_node_list_t **ptrlist = NULL;
791         be_node_list_t **ptrtmp;
792
793         if (pstart == NULL) /* Nothing to sort */
794             return (BE_SUCCESS);
795
796         /* Function: be_sort_list
797           * Description: Sort BE node list
798           * Parameters:
799           *   pointer to address of list head
800           *   compare function
801           *
802           * Return:
803           *   BE_SUCCESS - Success
804           *   be_errno_t - Failure
805           *
806           * Returns:
807           *   nothing
808           *
809           * Side effect:
810           *   node list sorted by name
811           *
812           * Scope:
813           *   Private
814           */
815
816     static int
817     static void
818     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
819     {
820         int ret = BE_SUCCESS;
821         size_t ibe, nbe;
822         be_node_list_t *p = NULL;
823         be_node_list_t **ptrlist = NULL;
824         be_node_list_t **ptrtmp;
825
826         if (pstart == NULL) /* Nothing to sort */
827             return (BE_SUCCESS);
828
829         /* Function: be_sort_list
830           * Description: Sort BE node list
831           * Parameters:
832           *   pointer to address of list head
833           *   compare function
834           *
835           * Return:
836           *   BE_SUCCESS - Success
837           *   be_errno_t - Failure
838           *
839           * Returns:
840           *   nothing
841           *
842           * Side effect:
843           *   node list sorted by name
844           *
845           * Scope:
846           *   Private
847           */
848
849     static int
850     static void
851     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
852     {
853         int ret = BE_SUCCESS;
854         size_t ibe, nbe;
855         be_node_list_t *p = NULL;
856         be_node_list_t **ptrlist = NULL;
857         be_node_list_t **ptrtmp;
858
859         if (pstart == NULL) /* Nothing to sort */
860             return (BE_SUCCESS);
861
862         /* Function: be_sort_list
863           * Description: Sort BE node list
864           * Parameters:
865           *   pointer to address of list head
866           *   compare function
867           *
868           * Return:
869           *   BE_SUCCESS - Success
870           *   be_errno_t - Failure
871           *
872           * Returns:
873           *   nothing
874           *
875           * Side effect:
876           *   node list sorted by name
877           *
878           * Scope:
879           *   Private
880           */
881
882     static int
883     static void
884     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
885     {
886         int ret = BE_SUCCESS;
887         size_t ibe, nbe;
888         be_node_list_t *p = NULL;
889         be_node_list_t **ptrlist = NULL;
890         be_node_list_t **ptrtmp;
891
892         if (pstart == NULL) /* Nothing to sort */
893             return (BE_SUCCESS);
894
895         /* Function: be_sort_list
896           * Description: Sort BE node list
897           * Parameters:
898           *   pointer to address of list head
899           *   compare function
900           *
901           * Return:
902           *   BE_SUCCESS - Success
903           *   be_errno_t - Failure
904           *
905           * Returns:
906           *   nothing
907           *
908           * Side effect:
909           *   node list sorted by name
910           *
911           * Scope:
912           *   Private
913           */
914
915     static int
916     static void
917     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
918     {
919         int ret = BE_SUCCESS;
920         size_t ibe, nbe;
921         be_node_list_t *p = NULL;
922         be_node_list_t **ptrlist = NULL;
923         be_node_list_t **ptrtmp;
924
925         if (pstart == NULL) /* Nothing to sort */
926             return (BE_SUCCESS);
927
928         /* Function: be_sort_list
929           * Description: Sort BE node list
930           * Parameters:
931           *   pointer to address of list head
932           *   compare function
933           *
934           * Return:
935           *   BE_SUCCESS - Success
936           *   be_errno_t - Failure
937           *
938           * Returns:
939           *   nothing
940           *
941           * Side effect:
942           *   node list sorted by name
943           *
944           * Scope:
945           *   Private
946           */
947
948     static int
949     static void
950     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
951     {
952         int ret = BE_SUCCESS;
953         size_t ibe, nbe;
954         be_node_list_t *p = NULL;
955         be_node_list_t **ptrlist = NULL;
956         be_node_list_t **ptrtmp;
957
958         if (pstart == NULL) /* Nothing to sort */
959             return (BE_SUCCESS);
960
961         /* Function: be_sort_list
962           * Description: Sort BE node list
963           * Parameters:
964           *   pointer to address of list head
965           *   compare function
966           *
967           * Return:
968           *   BE_SUCCESS - Success
969           *   be_errno_t - Failure
970           *
971           * Returns:
972           *   nothing
973           *
974           * Side effect:
975           *   node list sorted by name
976           *
977           * Scope:
978           *   Private
979           */
980
981     static int
982     static void
983     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
984     {
985         int ret = BE_SUCCESS;
986         size_t ibe, nbe;
987         be_node_list_t *p = NULL;
988         be_node_list_t **ptrlist = NULL;
989         be_node_list_t **ptrtmp;
990
991         if (pstart == NULL) /* Nothing to sort */
992             return (BE_SUCCESS);
993
994         /* Function: be_sort_list
995           * Description: Sort BE node list
996           * Parameters:
997           *   pointer to address of list head
998           *   compare function
999           *
1000           * Return:
1001           *   BE_SUCCESS - Success
1002           *   be_errno_t - Failure
1003           *
1004           * Returns:
1005           *   nothing
1006           *
1007           * Side effect:
1008           *   node list sorted by name
1009           *
1010           * Scope:
1011           *   Private
1012           */
1013
1014     static int
1015     static void
1016     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
1017     {
1018         int ret = BE_SUCCESS;
1019         size_t ibe, nbe;
1020         be_node_list_t *p = NULL;
1021         be_node_list_t **ptrlist = NULL;
1022         be_node_list_t **ptrtmp;
1023
1024         if (pstart == NULL) /* Nothing to sort */
1025             return (BE_SUCCESS);
1026
1027         /* Function: be_sort_list
1028           * Description: Sort BE node list
1029           * Parameters:
1030           *   pointer to address of list head
1031           *   compare function
1032           *
1033           * Return:
1034           *   BE_SUCCESS - Success
1035           *   be_errno_t - Failure
1036           *
1037           * Returns:
1038           *   nothing
1039           *
1040           * Side effect:
1041           *   node list sorted by name
1042           *
1043           * Scope:
1044           *   Private
1045           */
1046
1047     static int
1048     static void
1049     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
1050     {
1051         int ret = BE_SUCCESS;
1052         size_t ibe, nbe;
1053         be_node_list_t *p = NULL;
1054         be_node_list_t **ptrlist = NULL;
1055         be_node_list_t **ptrtmp;
1056
1057         if (pstart == NULL) /* Nothing to sort */
1058             return (BE_SUCCESS);
1059
1060         /* Function: be_sort_list
1061           * Description: Sort BE node list
1062           * Parameters:
1063           *   pointer to address of list head
1064           *   compare function
1065           *
1066           * Return:
1067           *   BE_SUCCESS - Success
1068           *   be_errno_t - Failure
1069           *
1070           * Returns:
1071           *   nothing
1072           *
1073           * Side effect:
1074           *   node list sorted by name
1075           *
1076           * Scope:
1077           *   Private
1078           */
1079
1080     static int
1081     static void
1082     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
1083     {
1084         int ret = BE_SUCCESS;
1085         size_t ibe, nbe;
1086         be_node_list_t *p = NULL;
1087         be_node_list_t **ptrlist = NULL;
1088         be_node_list_t **ptrtmp;
1089
1090         if (pstart == NULL) /* Nothing to sort */
1091             return (BE_SUCCESS);
1092
1093         /* Function: be_sort_list
1094           * Description: Sort BE node list
1095           * Parameters:
1096           *   pointer to address of list head
1097           *   compare function
1098           *
1099           * Return:
1100           *   BE_SUCCESS - Success
1101           *   be_errno_t - Failure
1102           *
1103           * Returns:
1104           *   nothing
1105           *
1106           * Side effect:
1107           *   node list sorted by name
1108           *
1109           * Scope:
1110           *   Private
1111           */
1112
1113     static int
1114     static void
1115     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
1116     {
1117         int ret = BE_SUCCESS;
1118         size_t ibe, nbe;
1119         be_node_list_t *p = NULL;
1120         be_node_list_t **ptrlist = NULL;
1121         be_node_list_t **ptrtmp;
1122
1123         if (pstart == NULL) /* Nothing to sort */
1124             return (BE_SUCCESS);
1125
1126         /* Function: be_sort_list
1127           * Description: Sort BE node list
1128           * Parameters:
1129           *   pointer to address of list head
1130           *   compare function
1131           *
1132           * Return:
1133           *   BE_SUCCESS - Success
1134           *   be_errno_t - Failure
1135           *
1136           * Returns:
1137           *   nothing
1138           *
1139           * Side effect:
1140           *   node list sorted by name
1141           *
1142           * Scope:
1143           *   Private
1144           */
1145
1146     static int
1147     static void
1148     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
1149     {
1150         int ret = BE_SUCCESS;
1151         size_t ibe, nbe;
1152         be_node_list_t *p = NULL;
1153         be_node_list_t **ptrlist = NULL;
1154         be_node_list_t **ptrtmp;
1155
1156         if (pstart == NULL) /* Nothing to sort */
1157             return (BE_SUCCESS);
1158
1159         /* Function: be_sort_list
1160           * Description: Sort BE node list
1161           * Parameters:
1162           *   pointer to address of list head
1163           *   compare function
1164           *
1165           * Return:
1166           *   BE_SUCCESS - Success
1167           *   be_errno_t - Failure
1168           *
1169           * Returns:
1170           *   nothing
1171           *
1172           * Side effect:
1173           *   node list sorted by name
1174           *
1175           * Scope:
1176           *   Private
1177           */
1178
1179     static int
1180     static void
1181     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
1182     {
1183         int ret = BE_SUCCESS;
1184         size_t ibe, nbe;
1185         be_node_list_t *p = NULL;
1186         be_node_list_t **ptrlist = NULL;
1187         be_node_list_t **ptrtmp;
1188
1189         if (pstart == NULL) /* Nothing to sort */
1190             return (BE_SUCCESS);
1191
1192         /* Function: be_sort_list
1193           * Description: Sort BE node list
1194           * Parameters:
1195           *   pointer to address of list head
1196           *   compare function
1197           *
1198           * Return:
1199           *   BE_SUCCESS - Success
1200           *   be_errno_t - Failure
1201           *
1202           * Returns:
1203           *   nothing
1204           *
1205           * Side effect:
1206           *   node list sorted by name
1207           *
1208           * Scope:
1209           *   Private
1210           */
1211
1212     static int
1213     static void
1214     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
1215     {
1216         int ret = BE_SUCCESS;
1217         size_t ibe, nbe;
1218         be_node_list_t *p = NULL;
1219         be_node_list_t **ptrlist = NULL;
1220         be_node_list_t **ptrtmp;
1221
1222         if (pstart == NULL) /* Nothing to sort */
1223             return (BE_SUCCESS);
1224
1225         /* Function: be_sort_list
1226           * Description: Sort BE node list
1227           * Parameters:
1228           *   pointer to address of list head
1229           *   compare function
1230           *
1231           * Return:
1232           *   BE_SUCCESS - Success
1233           *   be_errno_t - Failure
1234           *
1235           * Returns:
1236           *   nothing
1237           *
1238           * Side effect:
1239           *   node list sorted by name
1240           *
1241           * Scope:
1242           *   Private
1243           */
1244
1245     static int
1246     static void
1247     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
1248     {
1249         int ret = BE_SUCCESS;
1250         size_t ibe, nbe;
1251         be_node_list_t *p = NULL;
1252         be_node_list_t **ptrlist = NULL;
1253         be_node_list_t **ptrtmp;
1254
1255         if (pstart == NULL) /* Nothing to sort */
1256             return (BE_SUCCESS);
1257
1258         /* Function: be_sort_list
1259           * Description: Sort BE node list
1260           * Parameters:
1261           *   pointer to address of list head
1262           *   compare function
1263           *
1264           * Return:
1265           *   BE_SUCCESS - Success
1266           *   be_errno_t - Failure
1267           *
1268           * Returns:
1269           *   nothing
1270           *
1271           * Side effect:
1272           *   node list sorted by name
1273           *
1274           * Scope:
1275           *   Private
1276           */
1277
1278     static int
1279     static void
1280     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
1281     {
1282         int ret = BE_SUCCESS;
1283         size_t ibe, nbe;
1284         be_node_list_t *p = NULL;
1285         be_node_list_t **ptrlist = NULL;
1286         be_node_list_t **ptrtmp;
1287
1288         if (pstart == NULL) /* Nothing to sort */
1289             return (BE_SUCCESS);
1290
1291         /* Function: be_sort_list
1292           * Description: Sort BE node list
1293           * Parameters:
1294           *   pointer to address of list head
1295           *   compare function
1296           *
1297           * Return:
1298           *   BE_SUCCESS - Success
1299           *   be_errno_t - Failure
1300           *
1301           * Returns:
1302           *   nothing
1303           *
1304           * Side effect:
1305           *   node list sorted by name
1306           *
1307           * Scope:
1308           *   Private
1309           */
1310
1311     static int
1312     static void
1313     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
1314     {
1315         int ret = BE_SUCCESS;
1316         size_t ibe, nbe;
1317         be_node_list_t *p = NULL;
1318         be_node_list_t **ptrlist = NULL;
1319         be_node_list_t **ptrtmp;
1320
1321         if (pstart == NULL) /* Nothing to sort */
1322             return (BE_SUCCESS);
1323
1324         /* Function: be_sort_list
1325           * Description: Sort BE node list
1326           * Parameters:
1327           *   pointer to address of list head
1328           *   compare function
1329           *
1330           * Return:
1331           *   BE_SUCCESS - Success
1332           *   be_errno_t - Failure
1333           *
1334           * Returns:
1335           *   nothing
1336           *
1337           * Side effect:
1338           *   node list sorted by name
1339           *
1340           * Scope:
1341           *   Private
1342           */
1343
1344     static int
1345     static void
1346     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
1347     {
1348         int ret = BE_SUCCESS;
1349         size_t ibe, nbe;
1350         be_node_list_t *p = NULL;
1351         be_node_list_t **ptrlist = NULL;
1352         be_node_list_t **ptrtmp;
1353
1354         if (pstart == NULL) /* Nothing to sort */
1355             return (BE_SUCCESS);
1356
1357         /* Function: be_sort_list
1358           * Description: Sort BE node list
1359           * Parameters:
1360           *   pointer to address of list head
1361           *   compare function
1362           *
1363           * Return:
1364           *   BE_SUCCESS - Success
1365           *   be_errno_t - Failure
1366           *
1367           * Returns:
1368           *   nothing
1369           *
1370           * Side effect:
1371           *   node list sorted by name
1372           *
1373           * Scope:
1374           *   Private
1375           */
1376
1377     static int
1378     static void
1379     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
1380     {
1381         int ret = BE_SUCCESS;
1382         size_t ibe, nbe;
1383         be_node_list_t *p = NULL;
1384         be_node_list_t **ptrlist = NULL;
1385         be_node_list_t **ptrtmp;
1386
1387         if (pstart == NULL) /* Nothing to sort */
1388             return (BE_SUCCESS);
1389
1390         /* Function: be_sort_list
1391           * Description: Sort BE node list
1392           * Parameters:
1393           *   pointer to address of list head
1394           *   compare function
1395           *
1396           * Return:
1397           *   BE_SUCCESS - Success
1398           *   be_errno_t - Failure
1399           *
1400           * Returns:
1401           *   nothing
1402           *
1403           * Side effect:
1404           *   node list sorted by name
1405           *
1406           * Scope:
1407           *   Private
1408           */
1409
1410     static int
1411     static void
1412     be_sort_list(be_node_list_t **pstart, int (*compar)(const void *, const void *))
1413     {
1414         int ret = BE_SUCCESS;
1415         size_t ibe, nbe;
1416         be_node_list_t *p = NULL;
1417         be_node_list_t **ptrlist = NULL;
1418         be_node_list_t **ptrtmp;
1419
1420         if (pstart == NULL) /* Nothing to sort */
1421             return (BE_SUCCESS);
1422
1423         /* Function: be_sort_list
1424           * Description: Sort BE node list
1425           * Parameters:
1426           *   pointer to address of list head
1427           *   compare function
1428           *
1429           * Return:
1430           *   BE_SUCCESS - Success
1431           *   be_errno_t - Failure
1432           *
1433           * Returns:
1434           *   nothing
1435           *
1436           * Side effect:
1437           *   node list sorted by name

```

```
769     be_dataset_list_t ** const slist =
770         malloc(sizeof (be_dataset_list_t *) * (nmax + 1));
771     be_dataset_list_t *p;
772
773     if (slist == NULL) {
774         ret = BE_ERR_NOMEM;
775         if (slist == NULL)
776             continue;
777     }
778     /* build array of linked list dataset struct ptrs */
779     for (ns = 0, p = ptrlist[ibe]->be_node_datasets;
780          ns < nmax && p != NULL;
781          ns++, p = p->be_next_dataset) {
782         slist[ns] = p;
783     }
784     if (ns < 2) /* subordinate datasets < 2 - no sort */
785         goto end_dataset;
786     slist[ns] = NULL; /* add terminator */
787     /* in-place list quicksort using qsort(3C) */
788     qsort(slist, ns, sizeof (be_dataset_list_t *),
789           be_qsort_compare_datasets);
790     /* rewrite list pointer chain, including terminator */
791     ptrlist[ibe]->be_node_datasets = slist[0];
792     for (k = 0; k < ns; k++)
793         slist[k]->be_next_dataset = slist[k + 1];
794 end_dataset:
795     free(slist);
796 }
797 free:
798     free(ptrlist);
799     return (ret);
800 }
```

unchanged portion omitted

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

```
*****
8501 Mon Jun  8 20:19:23 2015
new/usr/src/lib/libbe/common/libbe.h
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) 2008, 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 #ifndef \_LIBBE\_H  
33 #define \_LIBBE\_H  
  
35 #include <libnvpair.h>  
36 #include <uuid/uuid.h>  
37 #include <libzfs.h>  
  
39 #ifdef \_\_cplusplus  
40 extern "C" {  
41 #endif  
  
43 #define BE\_ATTR\_ORIG\_BE\_NAME "orig\_be\_name"  
44 #define BE\_ATTR\_ORIG\_BE\_POOL "orig\_be\_pool"  
45 #define BE\_ATTR\_SNAP\_NAME "snap\_name"  
  
47 #define BE\_ATTR\_NEW\_BE\_NAME "new\_be\_name"  
48 #define BE\_ATTR\_NEW\_BE\_POOL "new\_be\_pool"  
49 #define BE\_ATTR\_NEW\_BE\_DESC "new\_be\_desc"  
50 #define BE\_ATTR\_POLICY "policy"  
51 #define BE\_ATTR\_ZFS\_PROPERTIES "zfs\_properties"  
  
53 #define BE\_ATTR\_FS\_NAMES "fs\_names"  
54 #define BE\_ATTR\_FS\_NUM "fs\_num"  
55 #define BE\_ATTR\_SHARED\_FS\_NAMES "shared\_fs\_names"  
56 #define BE\_ATTR\_SHARED\_FS\_NUM "shared\_fs\_num"  
  
58 #define BE\_ATTR\_MOUNTPOINT "mountpoint"  
59 #define BE\_ATTR\_MOUNT\_FLAGS "mount\_flags"  
60 #define BE\_ATTR\_UNMOUNT\_FLAGS "unmount\_flags"  
61 #define BE\_ATTR\_DESTROY\_FLAGS "destroy\_flags"

1

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

```
62 #define BE_ATTR_ROOT_DS          "root_ds"  
63 #define BE_ATTR_UUID_STR         "uuid_str"  
  
65 #define BE_ATTR_ACTIVE            "active"  
66 #define BE_ATTR_ACTIVE_ON_BOOT   "active_boot"  
67 #define BE_ATTR_GLOBAL_ACTIVE    "global_active"  
68 #define BE_ATTR_SPACE             "space_used"  
69 #define BE_ATTR_DATASET           "dataset"  
70 #define BE_ATTR_STATUS            "status"  
71 #define BE_ATTR_DATE              "date"  
72 #define BE_ATTR_MOUNTED          "mounted"  
  
74 /*  
75  * libbe error codes  
76 *  
77  * NOTE: there is a copy of this enum in beadm/messages.py. To keep these  
78  * in sync please make sure to add any new error messages at the end  
79  * of this enumeration.  
80 */  
81 enum {  
82     BE_SUCCESS = 0,  
83     BE_ERR_ACCESS = 4000, /* permission denied */  
84     BE_ERR_ACTIVATE_CURR, /* Activation of current BE failed */  
85     BE_ERR_AUTONAME, /* auto naming failed */  
86     BE_ERR_BE_NOENT, /* No such BE */  
87     BE_ERR_BUSY, /* mount busy */  
88     BE_ERR_CANCELED, /* operation canceled */  
89     BE_ERR_CLONE, /* BE clone failed */  
90     BE_ERR_COPY, /* BE copy failed */  
91     BE_ERR_CREATDS, /* dataset creation failed */  
92     BE_ERR_CURR_BE_NOT_FOUND, /* Can't find current BE */  
93     BE_ERR_DESTROY, /* failed to destroy BE or snapshot */  
94     BE_ERR_DEMOTE, /* BE demotion failed */  
95     BE_ERR_DSTYPE, /* invalid dataset type */  
96     BE_ERR_BE_EXISTS, /* BE exists */  
97     BE_ERR_INIT, /* be_zfs_init failed */  
98     BE_ERR_INTR, /* interrupted system call */  
99     BE_ERR_INVAL, /* invalid argument */  
100    BE_ERR_INVALPROP, /* invalid property for dataset */  
101    BE_ERR_INVALMOUNTPOINT, /* Unexpected mountpoint */  
102    BE_ERR_MOUNT, /* mount failed */  
103    BE_ERR_MOUNTED, /* already mounted */  
104    BE_ERR_NAMETOOLONG, /* name > BUFSIZ */  
105    BE_ERR_NOENT, /* Doesn't exist */  
106    BE_ERR_POOL_NOENT, /* No such pool */  
107    BE_ERR_NODEV, /* No such device */  
108    BE_ERR_NOTMOUNTED, /* File system not mounted */  
109    BE_ERR_NOMEM, /* not enough memory */  
110    BE_ERR_NONINHERIT, /* property is not inheritable for BE dataset */  
111    BE_ERR_NXIO, /* No such device or address */  
112    BE_ERR_NOSPC, /* No space on device */  
113    BE_ERR_NOTSUP, /* Operation not supported */  
114    BE_ERR_OPEN, /* open failed */  
115    BE_ERR_PERM, /* Not owner */  
116    BE_ERR_UNAVAIL, /* The BE is currently unavailable */  
117    BE_ERR_PROMOTE, /* BE promotion failed */  
118    BE_ERR_ROFS, /* read only file system */  
119    BE_ERR_READONLYDS, /* read only dataset */  
120    BE_ERR_READONLYPROP, /* read only property */  
121    BE_ERR_SS_EXISTS, /* snapshot exists */  
122    BE_ERR_SS_NOENT, /* No such snapshot */  
123    BE_ERR_UMOUNT, /* unmount failed */  
124    BE_ERR_UMOUNT_CURR_BE, /* Can't unmount current BE */  
125    BE_ERR_UMOUNT_SHARED, /* unmount of shared File System failed */  
126    BE_ERR_UNKNOWN, /* Unknown error */  
127    BE_ERR_ZFS, /* ZFS returned an error */
```

2

```
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_UNMOUNT_ZONEROOT, /* Failed to umount a zone root */
139     BE_ERR_ZONES_UNMOUNT, /* Unable to umount 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