

new/usr/src/cmd/projadd/Makefile

```
*****
1269 Thu Dec 15 19:51:09 2016
new/usr/src/cmd/projadd/Makefile
Want projadd, projdel and projmod in C.
*****
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 2006 Sun Microsystems, Inc. All rights reserved.
22 # Copyright 2009 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24 #
25 # cmd/projadd/Makefile
26 #

28 include ../Makefile.cmd

30 SUBDIRS= projadd projmod projdel projtest
30 PROGS= projadd projmod projdel
31 USRSBINPROGS= $(PROGS):%$(ROOTUSRSBIN)/%
32 POFILES= $(PROGS):%=.po

32 all := TARGET = all
33 install := TARGET = install
34 clean := TARGET = clean
35 clobber := TARGET = clobber
36 lint := TARGET = lint
37 _msg := TARGET = _msg
34 # No msg catalog here.
35 POFILE=

37 CLOBBERFILES += $(PROGS)

39 .KEEP_STATE:

41 all install lint clean clobber _msg: $(SUBDIRS)
41 all: $(PROGS)

43 $(SUBDIRS): FRC
44     @cd $@; pwd; $(MAKE) $(MFLAGS) $(TARGET)
43 install : all .WAIT $(USRSBINPROGS)

46 FRC:
45 clean lint:

47 _msg: $(MSGDOMAIN) $(POFILES)
48     $(CP) $(POFILES) $(MSGDOMAIN)
```

1

new/usr/src/cmd/projadd/Makefile

```
50 $(MSGDOMAIN):
51     $(INS.dir)
53 clobber: clean
54     $(RM) $(PROG) $(CLOBBERFILES)
56 $(ROOTUSRSBIN)/% : %
57     $(INS.file)
```

2

```
*****
```

```
1105 Thu Dec 15 19:51:09 2016
```

```
new/usr/src/cmd/projadd/Makefile.projadd
```

```
Want projadd, projdel and projmod in C.
```

```
*****
```

```
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 2009 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24 #
25 # cmd/stat/Makefile.stat

27 PROJADD = $(SRC)/cmd/projadd
28 PROJADDCOMMONDIR = $(PROJADD)/common

30 COMMON_OBJS = projet.o attrib.o util.o resctl.o lst.o
31 COMMON_SRCS = $(COMMON_OBJS:%.o=$(PROJADDCOMMONDIR)%.c)
```

```
*****
33008 Thu Dec 15 19:51:09 2016
new/usr/src/cmd/projadd/common/attrib.c
Want projadd, projdel and projmod in C.
*****
```

```

1 #include <sys/types.h>
2 #include <sys/stat.h>
3 #include <fcntl.h>
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <string.h>
7 #include <errno.h>
8 #include <locale.h>
9 #include <stddef.h>
10 #include <limits.h>
11 #include <rctl.h>
12 #include <regex.h>
13 #include <ctype.h>

15 #include <sys/debug.h>

17 #include "attrib.h"
18 #include "resctl.h"
19 #include "util.h"

21 #define BOSTR_REG_EXP   "^\n"
22 #define EOSTR_REG_EXP  "$"
23 #define EQUAL_REG_EXP  "="
24 #define STRNO_REG_EXP  "(.)"
25 #define IDENT_REG_EXP  "[[:alpha:]][[:alnum:]\_.-]*"
26 #define INTNM_REG_EXP  "[[:digit:]]+"
27 #define SIGAC_REG_EXP  "sig(nal)?(.*)?"
28 #define SIGHD_REG_EXP  "(signal|sig)"
29 #define SIGVL_REG_EXP  "(([[:digit:]]+)|((SIG)?([[:upper:]]+)([+-][123])?))"
30 #define SIGNL_REG_EXP  SIGHD_REG_EXP EQUAL_REG_EXP SIGVL_REG_EXP
31 #define STOCK_REG_EXP  "[[:upper:]]{1,5}([[:upper:]]{1,5})?,"?
32 #define ATTRB_REG_EXP  "(" STOCK_REG_EXP IDENT_REG_EXP ")"
33 #define ATVAL_REG_EXP  ATTRB_REG_EXP EQUAL_REG_EXP STRNO_REG_EXP

35 #define TO_EXP(X)        BOSTR_REG_EXP X EOSTR_REG_EXP

37 #define POOLN_EXP       TO_EXP(IDENT_REG_EXP)
38 #define INTNM_EXP       TO_EXP(INTNM_REG_EXP)
39 #define SIGAC_EXP       TO_EXP(SIGAC_REG_EXP)
40 #define SIGNL_EXP       TO_EXP(SIGNL_REG_EXP)
41 #define ATTRB_EXP       TO_EXP(ATTRB_REG_EXP)
42 #define ATVAL_EXP       TO_EXP(ATVAL_REG_EXP)

44 #define MAX_OF(X, Y)    (((X) > (Y)) ? (X) : (Y))

46 #define SEQU(X, Y)      (strcmp((X), (Y)) == 0)
47 #define SIN1(X, S1)     ((SEQU((X), (S1))))
48 #define SIN2(X, S1, S2) ((SEQU((X), (S1))) || (SIN1((X), (S2))))
49 #define SIN3(X, S1, S2, S3) ((SEQU((X), (S1))) || (SIN2((X), (S2)), (S3)))

51 #define ATT_VAL_TYPE_NULL 0
52 #define ATT_VAL_TYPE_VALUE 1
53 #define ATT_VAL_TYPE_LIST 2

55 #define ATT_ALLOC()      attrib_alloc()
56 #define ATT_VAL_ALLOC(T, V) attrib_val_alloc((T), (V))
57 #define ATT_VAL_ALLOC_NULL() ATT_VAL_ALLOC(ATT_VAL_TYPE_NULL, NULL)
58 #define ATT_VAL_ALLOC_VALUE(V) ATT_VAL_ALLOC(ATT_VAL_TYPE_VALUE, (V))
59 #define ATT_VAL_ALLOC_LIST(L) ATT_VAL_ALLOC(ATT_VAL_TYPE_LIST, (L))

61 typedef struct attrib_val_s {
```

```

62         int att_val_type;
63         /*LINTED*/
64         union {
65             char *att_val_value;
66             lst_t *att_val_values;
67         };
68 } attrib_val_t;

70 typedef struct attrib_s {
71     char *att_name;
72     attrib_val_t *att_value;
73 } attrib_t;

76 attrib_t *attrib_alloc();
77 attrib_val_t *attrib_val_alloc(int, void *);
78 char *attrib_val_tostring(attrib_val_t *, boolean_t);

80 int
81 attrib_validate_rctl(attrib_t *att, resctlrule_t *rule, lst_t *errlst)
82 {
83     int ret = 0;
84     char *atname = att->att_name;
85     attrib_val_t *atv, *atval = att->att_value;
86     attrib_val_t *priv, *val, *action;
87     char *vpriv, *vval, *vaction, *sigstr;
88     int atv_type = atval->att_val_type;
89     char *str;
90     int i, j, k;

92     int nmatch;
93     uint8_t rpriv, raction, sigval;
94     uint64_t rmax;
95     regex_t pintexp, sigacexp, signlexp;
96     regmatch_t *mat;

98     int nonecount, denycount, sigcount;

100    if (regcomp(&pintexp, INTNM_EXP, REG_EXTENDED) != 0) {
101        util_add_errmsg(errlst, gettext(
102            "Failed to compile regex for pos. integer"));
103        return (1);
104    }

106    if (regcomp(&sigacexp, SIGAC_EXP, REG_EXTENDED) != 0) {
107        util_add_errmsg(errlst, gettext(
108            "Failed to compile regex for sigaction"));
109        regfree(&pintexp);
110        return (1);
111    }

113    if (regcomp(&signlexp, SIGNL_EXP, REG_EXTENDED) != 0) {
114        util_add_errmsg(errlst, gettext(
115            "Failed to compile regex for signal"));
116        regfree(&pintexp);
117        regfree(&sigacexp);
118        return (1);
119    }

121    nmatch = signlexp.re_nsub + 1;
122    mat = util_safe_zmalloc(nmatch * sizeof (regmatch_t));

124    if (atv_type != ATT_VAL_TYPE_LIST) {
125        util_add_errmsg(errlst, gettext(
126            "rctl \"%s\" missing value"), atname));
127        ret = 1;
```

```

128         goto out;
129     }
130
131     for (i = 0; i < lst_size(atval->att_val_values); i++) {
132         atv = lst_at(atval->att_val_values, i);
133         if (atv->att_val_type != ATT_VAL_TYPE_LIST) {
134             if ((str = attrib_val_tostring(atv, B_FALSE)) != NULL) {
135                 util_add_errmsg(errlst, gettext(
136                     "rctl \"%s\" value \"%s\" "
137                     "should be in ()"), atname, str);
138                 free(str);
139             } else {
140                 util_add_errmsg(errlst, gettext(
141                     "rctl \"%s\" value "
142                     "should be in ()"), atname);
143             }
144             ret = 1;
145             continue;
146         }
147         /* Values should be in the form (priv, val, actions) */
148         if (lst_size(atv->att_val_values) < 3) {
149             util_add_errmsg(errlst, gettext(
150                 "rctl \"%s\" value should be in the form "
151                 "(priv, val, action[...])[,...]"), atname);
152             ret = 1;
153             continue;
154         }
155
156         priv = lst_at(atv->att_val_values, 0);
157         val = lst_at(atv->att_val_values, 1);
158         /* actions = el[2], el[3], ... */
159
160         vpriv = priv->att_val_value;
161         rpriv = rule->resctl_privs;
162
163         if (priv->att_val_type != ATT_VAL_TYPE_VALUE) {
164             util_add_errmsg(errlst, gettext(
165                 "rctl \"%s\" invalid privilege"), atname);
166             ret = 1;
167         } else if (!SIN3(vpriv, "basic", "privileged", "priv")) {
168             util_add_errmsg(errlst, gettext(
169                 "rctl \"%s\" unknown privilege \"%s\"", 
170                     atname, vpriv));
171             ret = 1;
172         } else if (!((
173             (rpriv & RESCTL_PRIV_PRIVE) &&
174             SEQU(vpriv, "priv")) ||
175             ((rpriv & RESCTL_PRIV_PRIVD) &&
176             SEQU(vpriv, "privileged")) ||
177             ((rpriv & RESCTL_PRIV_BASIC) &&
178             SEQU(vpriv, "basic")))) {
179             util_add_errmsg(errlst, gettext(
180                 "rctl \"%s\" privilege not allowed \"%s\"", 
181                     atname, vpriv));
182             ret = 1;
183         }
184
185         vval = val->att_val_value;
186         rmax = rule->resctl_max;
187
188         if (val->att_val_type != ATT_VAL_TYPE_VALUE) {
189             util_add_errmsg(errlst, gettext(
190                 "rctl \"%s\" invalid value"), atname);
191             ret = 1;
192         } else if (regexec(&pintexp, vval, 0, NULL, 0) != 0) {
193             util_add_errmsg(errlst, gettext(

```

```

194                     "rctl \"%s\" value \"%s\" is not an integer"),
195                     atname, vval);
196         ret = 1;
197     } else if (strtoll(vval, NULL, 0) > rmax) {
198         util_add_errmsg(errlst, gettext(
199             "rctl \"%s\" value \"%s\" exceeds system limit"),
200             atname, vval);
201         ret = 1;
202     }
203
204     nonecount = 0;
205     denycount = 0;
206     sigcount = 0;
207
208     for (j = 2; j < lst_size(atv->att_val_values); j++) {
209         action = lst_at(atv->att_val_values, j);
210
211         if (action->att_val_type != ATT_VAL_TYPE_VALUE) {
212             util_add_errmsg(errlst, gettext(
213                 "rctl \"%s\" invalid action"), atname);
214             ret = 1;
215             continue;
216         }
217
218         vaction = action->att_val_value;
219
220         if (regexec(&sigacexp, vaction, 0, NULL, 0) != 0 &&
221             !SIN2(vaction, "none", "deny")) {
222             util_add_errmsg(errlst, gettext(
223                 "rctl \"%s\" unknown action \"%s\"", 
224                     atname, vaction));
225             ret = 1;
226             continue;
227         }
228
229         raction = rule->resctl_action;
230         if (!((raction & RESCTL_ACTN_SIGN) &&
231             regexec(&sigacexp, vaction, 0, NULL, 0) == 0) ||
232             ((raction & RESCTL_ACTN_NONE) &&
233             SEQU(vaction, "none")) ||
234             ((raction & RESCTL_ACTN_DENY) &&
235             SEQU(vaction, "deny"))) {
236             util_add_errmsg(errlst, gettext(
237                 "rctl \"%s\" action not allowed \"%s\"", 
238                     atname, vaction));
239             ret = 1;
240             continue;
241         }
242
243         if (SEQU(vaction, "none")) {
244             if (nonecount >= 1) {
245                 util_add_errmsg(errlst, gettext(
246                     "rctl \"%s\" duplicate action "
247                     "none"), atname);
248             }
249             nonecount++;
250             continue;
251         }
252
253         if (SEQU(vaction, "deny")) {
254             if (denycount >= 1) {
255                 util_add_errmsg(errlst, gettext(
256                     "rctl \"%s\" duplicate action "
257                     "deny"), atname);
258             }
259             ret = 1;
260         }
261     }
262
263     if (ret) {
264         if (sigcount >= 1) {
265             util_add_errmsg(errlst, gettext(
266                 "rctl \"%s\" too many unknown actions"), 
267                     atname);
268         }
269     }
270
271     if (ret) {
272         if (nonecount >= 1) {
273             util_add_errmsg(errlst, gettext(
274                 "rctl \"%s\" too many duplicate actions"), 
275                     atname);
276         }
277     }
278
279     if (denycount >= 1) {
280         util_add_errmsg(errlst, gettext(
281             "rctl \"%s\" too many duplicate actions"), 
282             atname);
283     }
284
285     if (ret) {
286         if (sigcount >= 1) {
287             util_add_errmsg(errlst, gettext(
288                 "rctl \"%s\" too many unknown actions"), 
289                     atname);
290         }
291     }
292
293     if (ret) {
294         if (nonecount >= 1) {
295             util_add_errmsg(errlst, gettext(
296                 "rctl \"%s\" too many duplicate actions"), 
297                     atname);
298         }
299     }
300
301     if (denycount >= 1) {
302         util_add_errmsg(errlst, gettext(
303             "rctl \"%s\" too many duplicate actions"), 
304             atname);
305     }
306
307     if (ret) {
308         if (sigcount >= 1) {
309             util_add_errmsg(errlst, gettext(
310                 "rctl \"%s\" too many unknown actions"), 
311                     atname);
312         }
313     }
314
315     if (ret) {
316         if (nonecount >= 1) {
317             util_add_errmsg(errlst, gettext(
318                 "rctl \"%s\" too many duplicate actions"), 
319                     atname);
320         }
321     }
322
323     if (denycount >= 1) {
324         util_add_errmsg(errlst, gettext(
325             "rctl \"%s\" too many duplicate actions"), 
326             atname);
327     }
328
329     if (ret) {
330         if (sigcount >= 1) {
331             util_add_errmsg(errlst, gettext(
332                 "rctl \"%s\" too many unknown actions"), 
333                     atname);
334         }
335     }
336
337     if (ret) {
338         if (nonecount >= 1) {
339             util_add_errmsg(errlst, gettext(
340                 "rctl \"%s\" too many duplicate actions"), 
341                     atname);
342         }
343     }
344
345     if (denycount >= 1) {
346         util_add_errmsg(errlst, gettext(
347             "rctl \"%s\" too many duplicate actions"), 
348             atname);
349     }
350
351     if (ret) {
352         if (sigcount >= 1) {
353             util_add_errmsg(errlst, gettext(
354                 "rctl \"%s\" too many unknown actions"), 
355                     atname);
356         }
357     }
358
359     if (ret) {
360         if (nonecount >= 1) {
361             util_add_errmsg(errlst, gettext(
362                 "rctl \"%s\" too many duplicate actions"), 
363                     atname);
364         }
365     }
366
367     if (denycount >= 1) {
368         util_add_errmsg(errlst, gettext(
369             "rctl \"%s\" too many duplicate actions"), 
370             atname);
371     }
372
373     if (ret) {
374         if (sigcount >= 1) {
375             util_add_errmsg(errlst, gettext(
376                 "rctl \"%s\" too many unknown actions"), 
377                     atname);
378         }
379     }
380
381     if (ret) {
382         if (nonecount >= 1) {
383             util_add_errmsg(errlst, gettext(
384                 "rctl \"%s\" too many duplicate actions"), 
385                     atname);
386         }
387     }
388
389     if (denycount >= 1) {
390         util_add_errmsg(errlst, gettext(
391             "rctl \"%s\" too many duplicate actions"), 
392             atname);
393     }
394
395     if (ret) {
396         if (sigcount >= 1) {
397             util_add_errmsg(errlst, gettext(
398                 "rctl \"%s\" too many unknown actions"), 
399                     atname);
400         }
401     }
402
403     if (ret) {
404         if (nonecount >= 1) {
405             util_add_errmsg(errlst, gettext(
406                 "rctl \"%s\" too many duplicate actions"), 
407                     atname);
408         }
409     }
410
411     if (denycount >= 1) {
412         util_add_errmsg(errlst, gettext(
413             "rctl \"%s\" too many duplicate actions"), 
414             atname);
415     }
416
417     if (ret) {
418         if (sigcount >= 1) {
419             util_add_errmsg(errlst, gettext(
420                 "rctl \"%s\" too many unknown actions"), 
421                     atname);
422         }
423     }
424
425     if (ret) {
426         if (nonecount >= 1) {
427             util_add_errmsg(errlst, gettext(
428                 "rctl \"%s\" too many duplicate actions"), 
429                     atname);
430         }
431     }
432
433     if (denycount >= 1) {
434         util_add_errmsg(errlst, gettext(
435             "rctl \"%s\" too many duplicate actions"), 
436             atname);
437     }
438
439     if (ret) {
440         if (sigcount >= 1) {
441             util_add_errmsg(errlst, gettext(
442                 "rctl \"%s\" too many unknown actions"), 
443                     atname);
444         }
445     }
446
447     if (ret) {
448         if (nonecount >= 1) {
449             util_add_errmsg(errlst, gettext(
450                 "rctl \"%s\" too many duplicate actions"), 
451                     atname);
452         }
453     }
454
455     if (denycount >= 1) {
456         util_add_errmsg(errlst, gettext(
457             "rctl \"%s\" too many duplicate actions"), 
458             atname);
459     }
460
461     if (ret) {
462         if (sigcount >= 1) {
463             util_add_errmsg(errlst, gettext(
464                 "rctl \"%s\" too many unknown actions"), 
465                     atname);
466         }
467     }
468
469     if (ret) {
470         if (nonecount >= 1) {
471             util_add_errmsg(errlst, gettext(
472                 "rctl \"%s\" too many duplicate actions"), 
473                     atname);
474         }
475     }
476
477     if (denycount >= 1) {
478         util_add_errmsg(errlst, gettext(
479             "rctl \"%s\" too many duplicate actions"), 
480             atname);
481     }
482
483     if (ret) {
484         if (sigcount >= 1) {
485             util_add_errmsg(errlst, gettext(
486                 "rctl \"%s\" too many unknown actions"), 
487                     atname);
488         }
489     }
490
491     if (ret) {
492         if (nonecount >= 1) {
493             util_add_errmsg(errlst, gettext(
494                 "rctl \"%s\" too many duplicate actions"), 
495                     atname);
496         }
497     }
498
499     if (denycount >= 1) {
500         util_add_errmsg(errlst, gettext(
501             "rctl \"%s\" too many duplicate actions"), 
502             atname);
503     }
504
505     if (ret) {
506         if (sigcount >= 1) {
507             util_add_errmsg(errlst, gettext(
508                 "rctl \"%s\" too many unknown actions"), 
509                     atname);
510         }
511     }
512
513     if (ret) {
514         if (nonecount >= 1) {
515             util_add_errmsg(errlst, gettext(
516                 "rctl \"%s\" too many duplicate actions"), 
517                     atname);
518         }
519     }
520
521     if (denycount >= 1) {
522         util_add_errmsg(errlst, gettext(
523             "rctl \"%s\" too many duplicate actions"), 
524             atname);
525     }
526
527     if (ret) {
528         if (sigcount >= 1) {
529             util_add_errmsg(errlst, gettext(
530                 "rctl \"%s\" too many unknown actions"), 
531                     atname);
532         }
533     }
534
535     if (ret) {
536         if (nonecount >= 1) {
537             util_add_errmsg(errlst, gettext(
538                 "rctl \"%s\" too many duplicate actions"), 
539                     atname);
540         }
541     }
542
543     if (denycount >= 1) {
544         util_add_errmsg(errlst, gettext(
545             "rctl \"%s\" too many duplicate actions"), 
546             atname);
547     }
548
549     if (ret) {
550         if (sigcount >= 1) {
551             util_add_errmsg(errlst, gettext(
552                 "rctl \"%s\" too many unknown actions"), 
553                     atname);
554         }
555     }
556
557     if (ret) {
558         if (nonecount >= 1) {
559             util_add_errmsg(errlst, gettext(
560                 "rctl \"%s\" too many duplicate actions"), 
561                     atname);
562         }
563     }
564
565     if (denycount >= 1) {
566         util_add_errmsg(errlst, gettext(
567             "rctl \"%s\" too many duplicate actions"), 
568             atname);
569     }
570
571     if (ret) {
572         if (sigcount >= 1) {
573             util_add_errmsg(errlst, gettext(
574                 "rctl \"%s\" too many unknown actions"), 
575                     atname);
576         }
577     }
578
579     if (ret) {
580         if (nonecount >= 1) {
581             util_add_errmsg(errlst, gettext(
582                 "rctl \"%s\" too many duplicate actions"), 
583                     atname);
584         }
585     }
586
587     if (denycount >= 1) {
588         util_add_errmsg(errlst, gettext(
589             "rctl \"%s\" too many duplicate actions"), 
590             atname);
591     }
592
593     if (ret) {
594         if (sigcount >= 1) {
595             util_add_errmsg(errlst, gettext(
596                 "rctl \"%s\" too many unknown actions"), 
597                     atname);
598         }
599     }
599
600     if (ret) {
601         if (nonecount >= 1) {
602             util_add_errmsg(errlst, gettext(
603                 "rctl \"%s\" too many duplicate actions"), 
604                     atname);
605         }
606     }
607
608     if (denycount >= 1) {
609         util_add_errmsg(errlst, gettext(
610             "rctl \"%s\" too many duplicate actions"), 
611             atname);
612     }
613
614     if (ret) {
615         if (sigcount >= 1) {
616             util_add_errmsg(errlst, gettext(
617                 "rctl \"%s\" too many unknown actions"), 
618                     atname);
619         }
620     }
621
622     if (ret) {
623         if (nonecount >= 1) {
624             util_add_errmsg(errlst, gettext(
625                 "rctl \"%s\" too many duplicate actions"), 
626                     atname);
627         }
628     }
629
630     if (denycount >= 1) {
631         util_add_errmsg(errlst, gettext(
632             "rctl \"%s\" too many duplicate actions"), 
633             atname);
634     }
635
636     if (ret) {
637         if (sigcount >= 1) {
638             util_add_errmsg(errlst, gettext(
639                 "rctl \"%s\" too many unknown actions"), 
640                     atname);
641         }
642     }
643
644     if (ret) {
645         if (nonecount >= 1) {
646             util_add_errmsg(errlst, gettext(
647                 "rctl \"%s\" too many duplicate actions"), 
648                     atname);
649         }
650     }
651
652     if (denycount >= 1) {
653         util_add_errmsg(errlst, gettext(
654             "rctl \"%s\" too many duplicate actions"), 
655             atname);
656     }
657
658     if (ret) {
659         if (sigcount >= 1) {
660             util_add_errmsg(errlst, gettext(
661                 "rctl \"%s\" too many unknown actions"), 
662                     atname);
663         }
664     }
665
666     if (ret) {
667         if (nonecount >= 1) {
668             util_add_errmsg(errlst, gettext(
669                 "rctl \"%s\" too many duplicate actions"), 
670                     atname);
671         }
672     }
673
674     if (denycount >= 1) {
675         util_add_errmsg(errlst, gettext(
676             "rctl \"%s\" too many duplicate actions"), 
677             atname);
678     }
679
680     if (ret) {
681         if (sigcount >= 1) {
682             util_add_errmsg(errlst, gettext(
683                 "rctl \"%s\" too many unknown actions"), 
684                     atname);
685         }
686     }
687
688     if (ret) {
689         if (nonecount >= 1) {
690             util_add_errmsg(errlst, gettext(
691                 "rctl \"%s\" too many duplicate actions"), 
692                     atname);
693         }
694     }
695
696     if (denycount >= 1) {
697         util_add_errmsg(errlst, gettext(
698             "rctl \"%s\" too many duplicate actions"), 
699             atname);
700     }
701
702     if (ret) {
703         if (sigcount >= 1) {
704             util_add_errmsg(errlst, gettext(
705                 "rctl \"%s\" too many unknown actions"), 
706                     atname);
707         }
708     }
709
710     if (ret) {
711         if (nonecount >= 1) {
712             util_add_errmsg(errlst, gettext(
713                 "rctl \"%s\" too many duplicate actions"), 
714                     atname);
715         }
716     }
717
718     if (denycount >= 1) {
719         util_add_errmsg(errlst, gettext(
720             "rctl \"%s\" too many duplicate actions"), 
721             atname);
722     }
723
724     if (ret) {
725         if (sigcount >= 1) {
726             util_add_errmsg(errlst, gettext(
727                 "rctl \"%s\" too many unknown actions"), 
728                     atname);
729         }
730     }
731
732     if (ret) {
733         if (nonecount >= 1) {
734             util_add_errmsg(errlst, gettext(
735                 "rctl \"%s\" too many duplicate actions"), 
736                     atname);
737         }
738     }
739
740     if (denycount >= 1) {
741         util_add_errmsg(errlst, gettext(
742             "rctl \"%s\" too many duplicate actions"), 
743             atname);
744     }
745
746     if (ret) {
747         if (sigcount >= 1) {
748             util_add_errmsg(errlst, gettext(
749                 "rctl \"%s\" too many unknown actions"), 
750                     atname);
751         }
752     }
753
754     if (ret) {
755         if (nonecount >= 1) {
756             util_add_errmsg(errlst, gettext(
757                 "rctl \"%s\" too many duplicate actions"), 
758                     atname);
759         }
760     }
761
762     if (denycount >= 1) {
763         util_add_errmsg(errlst, gettext(
764             "rctl \"%s\" too many duplicate actions"), 
765             atname);
766     }
767
768     if (ret) {
769         if (sigcount >= 1) {
770             util_add_errmsg(errlst, gettext(
771                 "rctl \"%s\" too many unknown actions"), 
772                     atname);
773         }
774     }
775
776     if (ret) {
777         if (nonecount >= 1) {
778             util_add_errmsg(errlst, gettext(
779                 "rctl \"%s\" too many duplicate actions"), 
780                     atname);
781         }
782     }
783
784     if (denycount >= 1) {
785         util_add_errmsg(errlst, gettext(
786             "rctl \"%s\" too many duplicate actions"), 
787             atname);
788     }
789
790     if (ret) {
791         if (sigcount >= 1) {
792             util_add_errmsg(errlst, gettext(
793                 "rctl \"%s\" too many unknown actions"), 
794                     atname);
795         }
796     }
797
798     if (ret) {
799         if (nonecount >= 1) {
800             util_add_errmsg(errlst, gettext(
801                 "rctl \"%s\" too many duplicate actions"), 
802                     atname);
803         }
804     }
805
806     if (denycount >= 1) {
807         util_add_errmsg(errlst, gettext(
808             "rctl \"%s\" too many duplicate actions"), 
809             atname);
810     }
811
812     if (ret) {
813         if (sigcount >= 1) {
814             util_add_errmsg(errlst, gettext(
815                 "rctl \"%s\" too many unknown actions"), 
816                     atname);
817         }
818     }
819
820     if (ret) {
821         if (nonecount >= 1) {
822             util_add_errmsg(errlst, gettext(
823                 "rctl \"%s\" too many duplicate actions"), 
824                     atname);
825         }
826     }
827
828     if (denycount >= 1) {
829         util_add_errmsg(errlst, gettext(
830             "rctl \"%s\" too many duplicate actions"), 
831             atname);
832     }
833
834     if (ret) {
835         if (sigcount >= 1) {
836             util_add_errmsg(errlst, gettext(
837                 "rctl \"%s\" too many unknown actions"), 
838                     atname);
839         }
840     }
841
842     if (ret) {
843         if (nonecount >= 1) {
844             util_add_errmsg(errlst, gettext(
845                 "rctl \"%s\" too many duplicate actions"), 
846                     atname);
847         }
848     }
849
850     if (denycount >= 1) {
851         util_add_errmsg(errlst, gettext(
852             "rctl \"%s\" too many duplicate actions"), 
853             atname);
854     }
855
856     if (ret) {
857         if (sigcount >= 1) {
858             util_add_errmsg(errlst, gettext(
859                 "rctl \"%s\" too many unknown actions"), 
860                     atname);
861         }
862     }
863
864     if (ret) {
865         if (nonecount >= 1) {
866             util_add_errmsg(errlst, gettext(
867                 "rctl \"%s\" too many duplicate actions"), 
868                     atname);
869         }
870     }
871
872     if (denycount >= 1) {
873         util_add_errmsg(errlst, gettext(
874             "rctl \"%s\" too many duplicate actions"), 
875             atname);
876     }
877
878     if (ret) {
879         if (sigcount >= 1) {
880             util_add_errmsg(errlst, gettext(
881                 "rctl \"%s\" too many unknown actions"), 
882                     atname);
883         }
884     }
885
886     if (ret) {
887         if (nonecount >= 1) {
888             util_add_errmsg(errlst, gettext(
889                 "rctl \"%s\" too many duplicate actions"), 
890                     atname);
891         }
892     }
893
894     if (denycount >= 1) {
895         util_add_errmsg(errlst, gettext(
896             "rctl \"%s\" too many duplicate actions"), 
897             atname);
898     }
899
900     if (ret) {
901         if (sigcount >= 1) {
902             util_add_errmsg(errlst, gettext(
903                 "rctl \"%s\" too many unknown actions"), 
904                     atname);
905         }
906     }
907
908     if (ret) {
909         if (nonecount >= 1) {
910             util_add_errmsg(errlst, gettext(
911                 "rctl \"%s\" too many duplicate actions"), 
912                     atname);
913         }
914     }
915
916     if (denycount >= 1) {
917         util_add_errmsg(errlst, gettext(
918             "rctl \"%s\" too many duplicate actions"), 
919             atname);
920     }
921
922     if (ret) {
923         if (sigcount >= 1) {
924             util_add_errmsg(errlst, gettext(
925                 "rctl \"%s\" too many unknown actions"), 
926                     atname);
927         }
928     }
929
930     if (ret) {
931         if (nonecount >= 1) {
932             util_add_errmsg(errlst, gettext(
933                 "rctl \"%s\" too many duplicate actions"), 
934                     atname);
935         }
936     }
937
938     if (denycount >= 1) {
939         util_add_errmsg(errlst, gettext(
940             "rctl \"%s\" too many duplicate actions"), 
941             atname);
942     }
943
944     if (ret) {
945         if (sigcount >= 1) {
946             util_add_errmsg(errlst, gettext(
947                 "rctl \"%s\" too many unknown actions"), 
948                     atname);
949         }
950     }
951
952     if (ret) {
953         if (nonecount >= 1) {
954             util_add_errmsg(errlst, gettext(
955                 "rctl \"%s\" too many duplicate actions"), 
956                     atname);
957         }
958     }
959
960     if (denycount >= 1) {
961         util_add_errmsg(errlst, gettext(
962             "rctl \"%s\" too many duplicate actions"), 
963             atname);
964     }
965
966     if (ret) {
967         if (sigcount >= 1) {
968             util_add_errmsg(errlst, gettext(
969                 "rctl \"%s\" too many unknown actions"), 
970                     atname);
971         }
972     }
973
974     if (ret) {
975         if (nonecount >= 1) {
976             util_add_errmsg(errlst, gettext(
977                 "rctl \"%s\" too many duplicate actions"), 
978                     atname);
979         }
980     }
981
982     if (denycount >= 1) {
983         util_add_errmsg(errlst, gettext(
984             "rctl \"%s\" too many duplicate actions"), 
985             atname);
986     }
987
988     if (ret) {
989         if (sigcount >= 1) {
990             util_add_errmsg(errlst, gettext(
991                 "rctl \"%s\" too many unknown actions"), 
992                     atname);
993         }
994     }
995
996     if (ret) {
997         if (nonecount >= 1) {
998             util_add_errmsg(errlst, gettext(
999                 "rctl \"%s\" too many duplicate actions"), 
1000                    atname);
1001        }
1002    }
1003
1004    if (denycount >= 1) {
1005        util_add_errmsg(errlst, gettext(
1006            "rctl \"%s\" too many duplicate actions"), 
1007            atname);
1008    }
1009
1010    if (ret) {
1011        if (sigcount >= 1) {
1012            util_add_errmsg(errlst, gettext(
1013                "rctl \"%s\" too many unknown actions"), 
1014                    atname);
1015        }
1016    }
1017
1018    if (ret) {
1019        if (nonecount >= 1) {
1020            util_add_errmsg(errlst, gettext(
1021                "rctl \"%s\" too many duplicate actions"), 
1022                    atname);
1023        }
1024    }
1025
1026    if (denycount >= 1) {
1027        util_add_errmsg(errlst, gettext(
1028            "rctl \"%s\" too many duplicate actions"), 
1029            atname);
1030    }
1031
1032    if (ret) {
1033        if (sigcount >= 1) {
1034            util_add_errmsg(errlst, gettext(
1035                "rctl \"%s\" too many unknown actions"), 
1036                    atname);
1037        }
1038    }
1039
1040    if (ret) {
1041        if (nonecount >= 1) {
1042            util_add_errmsg(errlst, gettext(
1043                "rctl \"%s\" too many duplicate actions"), 
1044                    atname);
1045        }
1046    }
1047
1048    if (denycount >= 1) {
1049        util_add_errmsg(errlst, gettext(
1050            "rctl \"%s\" too many duplicate actions"), 
1051            atname);
1052    }
1053
1054    if (ret) {
1055        if (sigcount >= 1) {
1056            util_add_errmsg(errlst, gettext(
1057                "rctl \"%s\" too many unknown actions"), 
1058                    atname);
1059        }
1060    }
1061
1062    if (ret) {
1063        if (nonecount >= 1) {
1064            util_add_errmsg(errlst, gettext(
1065                "rctl \"%s\" too many duplicate actions"), 
1066                    atname);
1067        }
1068    }
1069
1070    if (denycount >= 1) {
1071        util_add_errmsg(errlst, gettext(
1072            "rctl \"%s\" too many duplicate actions"), 
1073            atname);
1074    }
1075
1076    if (ret) {
1077        if (sigcount >= 1) {
1078            util_add_errmsg(errlst, gettext(
1079                "rctl \"%s\" too many unknown actions"), 
1080                    atname);
1081        }
1082    }
1083
1084    if (ret) {
1085        if (nonecount >= 1) {
1086            util_add_errmsg(errlst, gettext(
1087                "rctl \"%s\" too many duplicate actions"), 
1088                    atname);
1089        }
1090    }
1091
1092    if (denycount >= 1) {
1093        util_add_errmsg(errlst, gettext(
1094            "rctl \"%s\" too many duplicate actions"), 
1095            atname);
1096    }
1097
1098    if (ret) {
1099        if (sigcount >= 1) {
1100            util_add_errmsg(errlst, gettext(
1101                "rctl \"%s\" too many unknown actions"), 
1102                    atname);
1103        }
1104    }
1105
1106    if (ret) {
1107        if (nonecount >= 1) {
1108            util_add_errmsg(errlst, gettext(
1109                "rctl \"%s\" too many duplicate actions"), 
1110                    atname);
1111        }
1112    }
1113
1114    if (denycount >= 1) {
1115        util_add_errmsg(errlst, gettext(
1116            "rctl \"%s\" too many duplicate actions"), 
1117            atname);
1118    }
1119
1120    if (ret) {
1121        if (sigcount >= 1) {
1122            util_add_errmsg(errlst, gettext(
1123                "rctl \"%s\" too many unknown actions"), 
1124                    atname);
1125        }
1126    }
1127
1128    if (ret) {
1129        if (nonecount >= 1) {
1130            util_add_errmsg(errlst, gettext(
1131                "rctl \"%s\" too many duplicate actions"), 
1132                    atname);
1133        }
1134    }
1135
1136    if (denycount >= 1) {
1137        util_add_errmsg(errlst, gettext(
1138            "rctl \"%s\" too many duplicate actions"), 
1139            atname);
1140    }
1141
1142    if (ret) {
1143        if (sigcount >= 1) {
1144            util_add_errmsg(errlst, gettext(
1145                "rctl \"%s\" too many unknown actions"), 
1146                    atname);
1147        }
1148    }
1149
1150    if (ret) {
1151        if (nonecount >= 1) {
1152            util_add_errmsg(errlst, gettext(
1153                "rctl \"%s\" too many duplicate actions"), 
1154                    atname);
1155        }
1156    }
1157
1158    if (denycount >= 1) {
1159        util_add_errmsg(errlst, gettext(
1160            "rctl \"%s\" too many duplicate actions"), 
1161            atname);
1162    }
1163
1164    if (ret) {
1165        if (sigcount >= 1) {
1166            util_add_errmsg(errlst, gettext(
1167                "rctl \"%s\" too many unknown actions"), 
1168                    atname);
1169        }
1170    }
1171
1172    if (ret) {
1173        if (nonecount >= 1) {
1174            util_add_errmsg(errlst, gettext(
1175                "rctl \"%s\" too many duplicate actions"), 
1176                    atname);
1177        }
1178    }
1179
1180    if (denycount >= 1) {
1181        util_add_errmsg(errlst, gettext(
1182            "rctl \"%s\" too many duplicate actions"), 
1183            atname);
1184    }
1185
1186    if (ret) {
1187        if (sigcount >= 1) {
1188            util_add_errmsg(errlst, gettext(
1189                "rctl \"%s\" too many unknown actions"), 
1190                    atname);
1191        }
1192    }
1193
1194    if (ret) {
1195        if (nonecount >= 1) {
1196            util_add_errmsg(errlst, gettext(
1197                "rctl \"%s\" too many duplicate actions"), 
1198                    atname);
1199        }
1200    }
1201
1202    if (denycount >= 1) {
1203        util_add_errmsg(errlst, gettext(
1204            "rctl \"%s\" too many duplicate actions"), 
1205            atname);
1206    }
1207
1208    if (ret) {
1209        if (sigcount >= 1) {
1210            util_add_errmsg(errlst, gettext(
1211                "rctl \"%s\" too many unknown actions"), 
1212                    atname);
1213        }
1214    }
1215
1216    if (ret) {
1217        if (nonecount >= 1) {
1218            util_add_errmsg(errlst, gettext(
1219                "rctl \"%s\" too many duplicate actions"), 
1220                    atname);
1221        }
1222    }
1223
1224    if (denycount >= 1) {
1225        util_add_errmsg(errlst, gettext(
1226            "rctl \"%s\" too many duplicate actions"), 
1227            atname);
1228    }

```

```

260
261
262
263     }
264     denycount++;
265     continue;
266   }
267
268   /* At this point, the action must be signal. */
269   if (sigcount >= 1) {
270     util_add_errmsg(errlst, gettext(
271       "rctl \"%s\" duplicate action sig"),
272       atname);
273     ret = 1;
274   }
275   sigcount++;
276
277   /*
278    * Make sure signal is correct format, on of:
279    * sig=##
280    * signal=##
281    * sig=SIGXXX
282    * signal=SIGXXX
283    * sig=XXX
284    * signal=XXX
285    */
286
287   if (regexec(&signlexp, vaction, nmatch, mat, 0) != 0 ||
288       (sigstr = util_substr(&signlexp, mat, vaction, 2))
289       == NULL) {
290     util_add_errmsg(errlst, gettext(
291       "rctl \"%s\" invalid signal \"%s\"", atname, vaction));
292     ret = 1;
293     continue;
294   }
295
296   /* Our version of sigstr =~ s/SIG// */
297   if (strchr(sigstr, 'S') != NULL)
298     sigstr = strstr(sigstr, "SIG") + 3;
299
300   sigval = 0;
301   for (k = 0; k < SIGS_CNT; k++) {
302     if (SEQU(sigstr[k].sig, sigstr))
303       sigval |= sigstr[k].mask;
304   }
305   free(sigstr);
306
307   if (sigval == 0) {
308     util_add_errmsg(errlst, gettext(
309       "rctl \"%s\" invalid signal \"%s\"", atname, vaction));
310     ret = 1;
311     continue;
312   }
313
314   if (!(rule->resctl_sigs)) {
315     util_add_errmsg(errlst, gettext(
316       "rctl \"%s\" signal not allowed \"%s\"", atname, vaction));
317     ret = 1;
318     continue;
319   }
320
321   if (nonecount > 0 && (denycount > 0 || sigcount > 0)) {
322     util_add_errmsg(errlst, gettext(
323       "rctl \"%s\" action \"none\" specified with "
324       "other actions"),
325       atname);

```

```

326           ret = 1;
327     }
328   }
329
330   out:
331   free(mat);
332   regfree(&signlexp);
333   regfree(&sigacexp);
334   regfree(&pintexp);
335   return (ret);
336 }

337 int
338 attrib_validate(attrib_t *att, lst_t *errlst)
339 {
340   int ret = 0;
341   char *atname = att->att_name;
342   attrib_val_t *atv = att->att_value;
343   int atv_type = atv->att_val_type;
344   char *str, *eptr;
345   long long ll;

346   resctl_info_t rinfo;
347   resctrule_t rrule;

348   regex_t poolnexp;
349   if (regcomp(&poolnexp, POOLN_EXP, REG_EXTENDED) != 0) {
350     util_add_errmsg(errlst, gettext(
351       "Failed to compile poolname regular expression:"));
352     return (1);
353   }

354   if (SEQU(atname, "task.final")) {
355     if (atv_type != ATT_VAL_TYPE_NULL) {
356       util_add_errmsg(errlst, gettext(
357         "task.final should not have value"));
358     }
359     ret = 1;
360   }
361
362   } else if (SEQU(atname, "rcap.max-rss")) {
363     if (atv_type == ATT_VAL_TYPE_NULL) {
364       util_add_errmsg(errlst, gettext(
365         "rcap.max-rss missing value"));
366     }
367     ret = 1;
368   } else if (atv_type == ATT_VAL_TYPE_LIST) {
369     util_add_errmsg(errlst, gettext(
370       "rcap.max-rss should have single value"));
371     ret = 1;
372   } else if (atv_type == ATT_VAL_TYPE_VALUE) {
373     if ((str = attrib_val_tostring(atv, B_FALSE)) != NULL) {
374       ll = strtoll(str, &eptr, 0);
375       if (*eptr != '\0') {
376         util_add_errmsg(errlst, gettext(
377           "rcap.max-rss is not an integer "
378           "value: \"%s\"", str));
379       }
380     }
381     ret = 1;
382   } else if (ll == LLONG_MIN && errno == ERANGE) {
383     util_add_errmsg(errlst, gettext(
384       "rcap.max-rss too small"));
385     ret = 1;
386   } else if (ll == LLONG_MAX && errno == ERANGE) {
387     util_add_errmsg(errlst, gettext(
388       "rcap.max-rss too large"));
389     ret = 1;
390   } else if (ll < 0) {
391     util_add_errmsg(errlst, gettext(
392       "rcap.max-rss should not have "));

```

```

392             "negative value: \"%s\"), str);
393         ret = 1;
394     }
395     free(str);
396 } else {
397     util_add_errmsg(errlst, gettext(
398         "rcap.max-rss has invalid value"));
399     ret = 1;
400 }
401 } else if (SEQU(atname, "project.pool")) {
402     if (atv_type == ATT_VAL_TYPE_NULL) {
403         util_add_errmsg(errlst, gettext(
404             "project.pool missing value"));
405         ret = 1;
406     } else if (atv_type == ATT_VAL_TYPE_LIST) {
407         util_add_errmsg(errlst, gettext(
408             "project.pool should have single value"));
409         ret = 1;
410     } else if (atv_type == ATT_VAL_TYPE_VALUE) {
411         if ((str = attrib_val_tostring(atv, B_FALSE)) != NULL) {
412             if (regexec(&poolnexp, str, 0, NULL, 0) != 0) {
413                 util_add_errmsg(errlst, gettext(
414                     "project.pool: invalid pool "
415                     "name \"%s\""), str);
416                 ret = 1;
417             } else if (resctl_pool_exist(str) != 0) {
418                 util_add_errmsg(errlst, gettext(
419                     "project.pool: pools not enabled "
420                     "or pool does not exist: \"%s\""),
421                     str);
422                 ret = 1;
423             }
424             free(str);
425         } else {
426             util_add_errmsg(errlst, gettext(
427                 "project.pool has invalid value "));
428             ret = 1;
429         }
430     }
431 } else if (resctl_get_info(atname, &rinfo) == 0) {
432     resctl_get_rule(&rinfo, &rrule);
433     if (attrib_validate_rctl(att, &rrule, errlst) != 0) {
434         ret = 1;
435     }
436 }
437 }

438 regfree(&poolnexp);
439 return (ret);
440
441 }

442 int
443 attrib_validate_lst(lst_t *attribs, lst_t *errlst)
444 {
445     int i, j;
446     attrib_t *att;
447     char **atnames, **atlast;
448     char *atname;
449     int ret = 0;

450     atlast = atnames = util_safe_zmalloc(
451         (lst_size(attribs) + 1) * sizeof (char *));
452     for (i = 0; i < lst_size(attribs); i++) {
453         att = lst_at(attribs, i);
454
455         /* Validate this attribute */

```

```

456         if (attrib_validate(att, errlst) != 0)
457             ret = 1;
458         /* Make sure it is not duplicated */
459         for (j = 0; (atname = atnames[j]) != NULL; j++) {
460             if (strcmp(atname, att->att_name) == 0) {
461                 util_add_errmsg(errlst, gettext(
462                     "Duplicate attributes \"%s\"", atname));
463                 ret = 1;
464             }
465         }
466     }
467 }
468 */
469 /* Add it to the attribute name to the
470 * temporary list if not found
471 */
472 if (atname == NULL) {
473     *atlast++ = att->att_name;
474 }
475 free(atnames);
476 return (ret);
477
478 attrib_t *
479 attrib_alloc()
480 {
481     return (util_safe_zmalloc(sizeof (attrib_t)));
482 }
483
484 attrib_val_t *
485 attrib_val_alloc(int type, void *val)
486 {
487     attrib_val_t *ret;
488
489     attrib_val_t *ret;
490     ret = util_safe_malloc(sizeof (attrib_val_t));
491     ret->att_val_type = type;
492     ret->att_val_value = val;
493     return (ret);
494 }
495
496 char *
497 attrib_val_tostring(attrib_val_t *val, boolean_t innerlist)
498 {
499     char *ret = NULL;
500     char *vstring;
501     int i;
502     attrib_val_t *v;
503     switch (val->att_val_type) {
504         case ATT_VAL_TYPE_NULL:
505             return (util_safe_strdup(""));
506         case ATT_VAL_TYPE_VALUE:
507             return (util_safe_strdup(val->att_val_value));
508         case ATT_VAL_TYPE_LIST:
509             /* Only innerlists need to be betweenen ( and ) */
510             if (innerlist)
511                 ret = UTIL_STR_APPEND1(ret, "(");
512             for (i = 0; i < lst_size(val->att_val_values);
513                 i++) {
514                 v = lst_at(val->att_val_values, i);
515                 if (i > 0) {
516                     ret = UTIL_STR_APPEND1(ret, ",");
517                 }
518                 if ((vstring =
519                     attrib_val_tostring(v, B_TRUE)) == NULL) {
520                     UTIL_FREE_SNLL(ret);
521                     goto out;
522                 }
523             }

```

```

524             ret = UTIL_STR_APPEND1(ret, vstring);
525             free(vstring);
526         }
527         if (innerlist)
528             ret = UTIL_STR_APPEND1(ret, ")");
529     return (ret);
530 }

532 out:
533     return (ret);
534 }

536 char *
537 attrib_tostring(void *at)
538 {
539     attrib_t *att;
540     char *ret = NULL, *vstring;
541
542     att = (attrib_t *)at;
543     ret = UTIL_STR_APPEND1(ret, att->att_name);
544     if ((vstring = attrib_val_tostring(att->att_value, B_FALSE)) != NULL) {
545         if (strlen(vstring) > 0)
546             ret = UTIL_STR_APPEND2(ret, "=", vstring);
547         free(vstring);
548     }
549     return (ret);
550 }
551 UTIL_FREE_SNLL(ret);
552 return (ret);

554 char *
555 attrib_lst_tostring(lst_t *attrs)
556 {
557     int i;
558     attrib_t *att;
559     char *ret = NULL;
560     char *str;
561
562     ret = UTIL_STR_APPEND1(ret, "");
563     for (i = 0; i < lst_size(attrs); i++) {
564         att = lst_at(attrs, i);
565
566         if ((str = attrib_tostring(att)) != NULL) {
567             if (i > 0)
568                 ret = UTIL_STR_APPEND1(ret, ";");
569             ret = UTIL_STR_APPEND1(ret, str);
570             free(str);
571             continue;
572         }
573
574         free(ret);
575     }
576     return (NULL);
577 }
578 return (ret);

581 void
582 attrib_val_free(attrib_val_t *atv)
583 {
584     attrib_val_t *val;
585
586     if (atv->att_val_type == ATT_VAL_TYPE_VALUE) {
587         free(atv->att_val_value);
588     } else if (atv->att_val_type == ATT_VAL_TYPE_LIST) {
589         while (!lst_is_empty(atv->att_val_values)) {

```

```

590             val = lst_at(atv->att_val_values, 0);
591             (void) lst_remove(atv->att_val_values, val);
592             attrib_val_free(val);
593             free(val);
594         }
595     }
596     free(atv->att_val_values);
597 }

599 void
600 attrib_free(attrib_t *att)
601 {
602     free(att->att_name);
603     if (att->att_value != NULL) {
604         attrib_val_free(att->att_value);
605         free(att->att_value);
606     }
607 }
608 void
609 attrib_free_lst(lst_t *attribs)
610 {
611     attrib_t *att;
612
613     if (attribs == NULL)
614         return;
615
616     while (!lst_is_empty(attribs)) {
617         att = lst_at(attribs, 0);
618         (void) lst_remove(attribs, att);
619         attrib_free(att);
620         free(att);
621     }
622 }

624 void
625 attrib_sort_lst(lst_t *attribs)
626 {
627     int i, j, n;
628     attrib_t *atti, *attj;
629
630     if (attribs == NULL)
631         return;
632
633     n = lst_size(attribs);
634     for (i = 0; i < n - 1; i++) {
635         for (j = i + 1; j < n; j++) {
636             atti = lst_at(attribs, i);
637             attj = lst_at(attribs, j);
638             if (strcmp(attj->att_name, atti->att_name) < 0) {
639                 (void) lst_replace_at(attribs, i, attj);
640                 (void) lst_replace_at(attribs, j, atti);
641             }
642         }
643     }
644 }

646 void
647 attrib_val_to_list(attrib_val_t *atv)
648 {
649     void *val;
650     int type;
651     attrib_val_t *mat;
652
653     if (atv->att_val_type == ATT_VAL_TYPE_LIST)
654         return;

```

```

656     val = atv->att_val_value;
657     type = atv->att_val_type;
658
659     atv->att_val_type = ATT_VAL_TYPE_LIST;
660     atv->att_val_values = util_safe_malloc(sizeof (lst_t));
661     lst_create(atv->att_val_values);
662
663     if (type == ATT_VAL_TYPE_VALUE && val != NULL) {
664         mat = ATT_VAL_ALLOC_VALUE(val);
665         lst_insert_tail(atv->att_val_values, mat);
666     }
667 }
668
669 void
670 attrib_val_append(attrib_val_t *atv, char *token)
671 {
672     attrib_val_t *nat;
673     if (atv->att_val_type == ATT_VAL_TYPE_VALUE) {
674         /* convert this to LIST attribute */
675         attrib_val_to_list(atv);
676     }
677
678     if (atv->att_val_type == ATT_VAL_TYPE_NULL) {
679         /* convert this to VALUE attribute */
680         atv->att_val_type = ATT_VAL_TYPE_VALUE;
681         atv->att_val_value = util_safe_strdup(token);
682     } else if (atv->att_val_type == ATT_VAL_TYPE_LIST) {
683         /* append token to the list */
684         nat = ATT_VAL_ALLOC_VALUE(util_safe_strdup(token));
685         lst_insert_tail(atv->att_val_values, nat);
686     }
687 }
688
689 attrib_val_t *
690 attrib_val_parse(char *values, lst_t *errlst)
691 {
692     attrib_val_t *ret = NULL;
693     attrib_val_t *at;
694     attrib_val_t *nat;
695     lst_t stk;
696
697     char **tokens, *token, *usedtokens, *prev;
698     int i, error, parendepth;
699
700     error = parendepth = 0;
701     prev = "";
702
703     if ((tokens = util_tokenize(values, errlst)) == NULL) {
704         goto out1;
705     }
706
707     lst_create(&stk);
708     usedtokens = UTIL_STR_APPEND1(NULL, "");
709
710     at = ret = ATT_VAL_ALLOC_NULL();
711
712     for (i = 0; (token = tokens[i]) != NULL; i++) {
713
714         usedtokens = UTIL_STR_APPEND1(usedtokens, token);
715
716         if (SEQU(token, "")) {
717             if (SIN3(prev, "", "(, \"")) {
718                 attrib_val_append(at, "");
719             }
720             attrib_val_to_list(at);
721             prev = ",";
722         }
723     }
724
725     if (!SIN3(prev, "", "(, \"")) {
726         util_add_errmsg(errlst, gettext(
727             "\\""%s\" <- \"(\\" unexpected", usedtokens));
728         error = 1;
729         goto out;
730     }
731
732     switch (at->att_val_type) {
733         case ATT_VAL_TYPE_VALUE:
734             util_add_errmsg(errlst, gettext(
735                 "\\""%s\" <- \"%s\" unexpected"),
736                 usedtokens, token);
737             error = 1;
738             goto out;
739         case ATT_VAL_TYPE_NULL:
740             /* Make is a LIST attrib */
741             attrib_val_to_list(at);
742             /* FALLTHROUGH */
743         case ATT_VAL_TYPE_LIST:
744             /* Allocate NULL node */
745             nat = ATT_VAL_ALLOC_NULL();
746             attrib_val_to_list(nat);
747             lst_insert_tail(
748                 at->att_val_values, nat);
749             /* push at down one level */
750             lst_insert_head(&stk, at);
751             at = nat;
752             break;
753             parendepth++;
754             prev = "(";
755     } else if (SEQU(token, "))")) {
756         if (parendepth <= 0) {
757             util_add_errmsg(errlst, gettext(
758                 "\\""%s\" <- \")\" unexpected"), usedtokens);
759             error = 1;
760             goto out;
761         }
762         if (SIN2(prev, "", "((")) {
763             attrib_val_append(at, "");
764         }
765         if (!lst_is_empty(&stk)) {
766             at = lst_at(&stk, 0);
767             (void) lst_remove(&stk, at);
768         }
769         parendepth--;
770         prev = ")";
771     } else {
772         if (!SIN3(prev, "", "(, \"")) {
773             util_add_errmsg(errlst, gettext(
774                 "\\""%s\" <- \"%s\" unexpected"),
775                 usedtokens, token);
776             error = 1;
777             goto out;
778         }
779
780         attrib_val_append(at, token);
781         prev = token;
782     }
783 }
784
785 if (parendepth != 0) {
786     util_add_errmsg(errlst, gettext(
787         "\\""%s\" <- \")\" missing"),
788

```

```

788         usedtokens);
789         error = 1;
790         goto out;
791     }
792
793     if (SIN2(prev, ",", "")) {
794         switch (at->att_val_type) {
795             case ATT_VAL_TYPE_NULL:
796                 util_add_errmsg(errlst, gettext(
797                     "\'%s\' unexpected"),
798                     usedtokens);
799                 error = 1;
800                 goto out;
801             case ATT_VAL_TYPE_VALUE:
802             case ATT_VAL_TYPE_LIST:
803                 attrib_val_append(at, "");
804                 break;
805         }
806     }
807
808 out:
809     while (!lst_is_empty(&stk)) {
810         at = lst_at(&stk, 0);
811         (void) lst_remove(&stk, at);
812     }
813
814     util_free_tokens(tokens);
815     free(tokens);
816     free(usedtokens);
817     if (error) {
818         attrib_val_free(ret);
819         UTIL_FREE_SNLL(ret);
820     }
821 out1:
822     return (ret);
823 }
824
825 attrib_t *
826 attrib_parse(regex_t *attrbexp, regex_t *atvalexp, char *att, int flags,
827 lst_t *errlst)
828 {
829     int nmatch = MAX_OF(attrbexp->re_nsub, atvalexp->re_nsub) + 1;
830     attrib_t *ret = NULL;
831     attrib_val_t *retv, *atv, *atvl;
832     char *values = NULL;
833     int vidx, nidx, vlen;
834     int scale;
835
836     char *num, *mod, *unit;
837     int i;
838
839     resctl_info_t rinfo;
840     resctlrule_t rrule;
841
842     regmatch_t *mat = util_safe_malloc(nmatch * sizeof (regmatch_t));
843     ret = ATT_ALLOC();
844
845     if (regexec(attrbexp, att, attrbexp->re_nsub + 1, mat, 0) == 0) {
846         ret->att_name = util_safe_strdup(att);
847         ret->att_value = ATT_VAL_ALLOC_NULL();
848     } else if (regexec(atvalexp, att,
849         atvalexp->re_nsub + 1, mat, 0) == 0) {
850         vidx = atvalexp->re_nsub;
851         vlen = mat[vidx].rm_eo - mat[vidx].rm_so;
852         nidx = atvalexp->re_nsub - 3;
853         ret->att_name = util_substr(atvalexp, mat, att, nidx);

```

```

855         if (vlen > 0) {
856             values = util_substr(atvalexp, mat, att, vidx);
857             ret->att_value = attrib_val_parse(values, errlst);
858             free(values);
859             if (ret->att_value == NULL) {
860                 util_add_errmsg(errlst, gettext(
861                     "Invalid value on attribute \'%s\'"),
862                     ret->att_name);
863                 attrib_free(ret);
864                 UTIL_FREE_SNLL(ret);
865                 goto out;
866             }
867             /* the value is an empty string */
868             ret->att_value = ATT_VAL_ALLOC_NULL();
869         }
870     } else {
871         util_add_errmsg(errlst, gettext(
872             "Invalid attribute \'%s\'"), att);
873         attrib_free(ret);
874         UTIL_FREE_SNLL(ret);
875         goto out;
876     }
877 }
878
879 if (!(flags & F_PAR_UNT))
880     goto out;
881
882 if (SEQU(ret->att_name, "rcap.max-rss")) {
883     values = attrib_val_tostring(ret->att_value, B_FALSE);
884     if (util_val2num(values, BYTES_SCALE, errlst,
885         &num, &mod, &unit) == 0) {
886         attrib_val_free(ret->att_value);
887         ret->att_value = ATT_VAL_ALLOC_VALUE(num);
888         free(mod);
889         free(unit);
890     } else {
891         attrib_free(ret);
892         UTIL_FREE_SNLL(ret);
893         goto out;
894     }
895     free(values);
896 }
897
898 if (resctl_get_info(ret->att_name, &rinfo) == 0) {
899     resctl_get_rule(&rinfo, &rrule);
900     retv = ret->att_value;
901
902     switch (rrule.resctl_type) {
903         case RESCTL_TYPE_BYTES:
904             scale = BYTES_SCALE;
905             break;
906         case RESCTL_TYPE_SCNDS:
907             scale = SCNDS_SCALE;
908             break;
909         case RESCTL_TYPE_COUNT:
910             scale = COUNT_SCALE;
911             break;
912         default:
913             scale = UNKWN_SCALE;
914             break;
915     }
916
917     if (retv->att_val_type != ATT_VAL_TYPE_LIST)
918         goto out;

```

```

921     for (i = 0; i < lst_size(retv->att_val_values); i++) {
922         atvl = atv = lst_at(retv->att_val_values, i);
923
924         /*
925          * Continue if not a list and the second value
926          * is not a scalar value
927          */
928         if (atv->att_val_type != ATT_VAL_TYPE_LIST ||
929             lst_size(atv->att_val_values) < 3 ||
930             (atv = lst_at(atv->att_val_values, 1)) == NULL ||
931             atv->att_val_type != ATT_VAL_TYPE_VALUE) {
932             continue;
933         }
934         values = attrib_val_tostring(atv, B_FALSE);
935         if (util_val2num(values, scale, errlst,
936             &num, &mod, &unit) == 0) {
937             attrib_val_free(atv);
938             atv = ATT_VAL_ALLOC_VALUE(num);
939             (void) lst_replace_at(atvl->att_val_values, 1,
940                 atv);
941             free(mod);
942             free(unit);
943
944         } else {
945             free(values);
946             attrib_free(ret);
947             UTIL_FREE_SNNULL(ret);
948             goto out;
949         }
950     }
951     free(values);
952 }
953
954 out:
955     free(mat);
956     return (ret);
957 }
958
959 lst_t *
960 attrib_parse_attributes(char *attribs, int flags, lst_t *errlst)
961 {
962     char *sattrss, *atrrs, *att;
963     regex_t attrbexp, atvalexp;
964
965     attrib_t *natt = NULL;
966     lst_t *ret = NULL;
967
968     ret = util_safe_malloc(sizeof (lst_t));
969     lst_create(ret);
970
971     if (regcomp(&attrbexp, ATTRB_EXP, REG_EXTENDED) != 0)
972         goto out1;
973     if (regcomp(&atvalexp, ATVAL_EXP, REG_EXTENDED) != 0)
974         goto out2;
975
976     sattrss = attrrs = util_safe_strdup(attribs);
977     while ((att = strsep(&sattrss, ";")) != NULL) {
978         if (*att == '\0')
979             continue;
980         if ((natt = attrib_parse(&attrbexp,
981             &atvalexp, att, flags, errlst)) == NULL) {
982             attrib_free_lst(ret);
983             UTIL_FREE_SNNULL(ret);
984             break;
985     }

```

```

986         lst_insert_tail(ret, natt);
987     }
988
989     free(sattrss);
990     regfree(&atvalexp);
991     out2:
992     regfree(&attrbexp);
993     out1:
994     return (ret);
995 }
996
997 attrib_val_t *
998 attrib_val_duplicate(attrib_val_t *atv)
999 {
1000     int i;
1001     lst_t *values;
1002     attrib_val_t *val;
1003     attrib_val_t *natv;
1004
1005     switch (atv->att_val_type) {
1006         case ATT_VAL_TYPE_NULL:
1007             natv = ATT_VAL_ALLOC_NULL();
1008             break;
1009         case ATT_VAL_TYPE_VALUE:
1010             natv = ATT_VAL_ALLOC_VALUE(
1011                 util_safe_strdup(atv->att_val_value));
1012             break;
1013         case ATT_VAL_TYPE_LIST:
1014             values = util_safe_malloc(sizeof (lst_t));
1015             lst_create(values);
1016             for (i = 0; i < lst_size(atv->att_val_values); i++)
1017                 {
1018                     val = lst_at(atv->att_val_values, i);
1019                     lst_insert_tail(values,
1020                         attrib_val_duplicate(val));
1021                 }
1022             natv = ATT_VAL_ALLOC_LIST(values);
1023             break;
1024     }
1025
1026     return (natv);
1027 }
1028
1029 attrib_t *
1030 attrib_duplicate(attrib_t *att)
1031 {
1032     attrib_t *natt = ATT_ALLOC();
1033     natt->att_name = util_safe_strdup(att->att_name);
1034     natt->att_value = attrib_val_duplicate(att->att_value);
1035     return (natt);
1036 }
1037
1038 attrib_t *
1039 attrib_merge_add(attrib_t *eatt, attrib_t *natt)
1040 {
1041     int i;
1042     attrib_t *att;
1043     attrib_val_t *atv, *eatv, *natv;
1044     lst_t *values;
1045
1046     eatv = eatt->att_value;
1047     natv = natt->att_value;
1048     att = ATT_ALLOC();
1049     att->att_name = util_safe_strdup(eatt->att_name);
1050

```

new/usr/src/cmd/projadd/common/attrib.c

17

```

1052     if (eatv->att_val_type == ATT_VAL_TYPE_NULL) {
1053         /* NULL + X -> X */
1054         atv = attrib_val_duplicate(natv);
1055     } else if (natv->att_val_type == ATT_VAL_TYPE_NULL) {
1056         /* X + NULL -> X */
1057         atv = attrib_val_duplicate(eatv);
1058     } else if (eatv->att_val_type == ATT_VAL_TYPE_VALUE &&
1059     natv->att_val_type == ATT_VAL_TYPE_VALUE) {
1060
1061         /* VALUE + VALUE -> LIST */
1062         values = util_safe_malloc(sizeof (lst_t));
1063         lst_create(values);
1064         lst_insert_tail(values, attrib_val_duplicate(eatv));
1065         lst_insert_tail(values, attrib_val_duplicate(natv));
1066         atv = ATT_VAL_ALLOC_LIST(values);
1067
1068     } else if (eatv->att_val_type == ATT_VAL_TYPE_VALUE &&
1069     natv->att_val_type == ATT_VAL_TYPE_LIST) {
1070
1071         /* VALUE + LIST -> LIST */
1072         atv = attrib_val_duplicate(natv);
1073         lst_insert_head(atv->att_val_values,
1074                         attrib_val_duplicate(eatv));
1075
1076     } else if (eatv->att_val_type == ATT_VAL_TYPE_LIST &&
1077     natv->att_val_type == ATT_VAL_TYPE_VALUE) {
1078
1079         /* LIST + VALUE -> LIST */
1080         atv = attrib_val_duplicate(eatv);
1081         lst_insert_tail(atv->att_val_values,
1082                         attrib_val_duplicate(natv));
1083
1084     } else if (eatv->att_val_type == ATT_VAL_TYPE_LIST &&
1085     natv->att_val_type == ATT_VAL_TYPE_LIST) {
1086
1087         /* LIST + LIST -> LIST */
1088         atv = attrib_val_duplicate(eatv);
1089         for (i = 0; i < lst_size(natv->att_val_values); i++) {
1090             lst_insert_tail(atv->att_val_values,
1091                             attrib_val_duplicate(
1092                             lst_at(natv->att_val_values, i)));
1093         }
1094     }
1095
1096     att->att_value = atv;
1097     return (att);
1098 }
1099
1100 int
1101 attrib_val_equal(attrib_val_t *xatv, attrib_val_t *yatv)
1102 {
1103     int i;
1104     attrib_val_t *xv, *yv;
1105
1106     if (xatv->att_val_type != yatv->att_val_type)
1107         return (1);
1108
1109     switch (xatv->att_val_type) {
1110         case ATT_VAL_TYPE_NULL:
1111             return (0);
1112         case ATT_VAL_TYPE_VALUE:
1113             if (SEQU(xatv->att_val_value, yatv->att_val_value))
1114                 return (1);
1115             break;
1116         case ATT_VAL_TYPE_LIST:
1117             if (lst_size(xatv->att_val_values) != lst_size(yatv->att_val_values))
1118                 return (1);
1119             for (i = 0; i < lst_size(xatv->att_val_values); i++) {
1120                 xv = lst_at(xatv->att_val_values, i);
1121                 yv = lst_at(yatv->att_val_values, i);
1122                 if (!attrib_val_equal(xv, yv))
1123                     return (1);
1124             }
1125             break;
1126     }
1127     return (0);
1128 }
```

new/usr/src/cmd/projadd/common/attrib.c 18

```

1118                     return (0);
1119                 }
1120             }
1121         case ATT_VAL_TYPE_LIST:
1122             if (lst_size(xatv->att_val_values) != 
1123                 lst_size(yatv->att_val_values))
1124                 return (1);
1125             for (i = 0; i < lst_size(xatv->att_val_values);
1126                 i++) {
1127                 xv = lst_at(xatv->att_val_values, i);
1128                 yv = lst_at(yatv->att_val_values, i);
1129                 if (attrib_val_equal(xv, yv) != 0) {
1130                     return (1);
1131                 }
1132             }
1133         break;
1134     }
1135     return (0);
1136 }

1138 attrib_t *
1139 attrib_merge_remove(attrib_t *eatt, attrib_t *natt, lst_t *errlst)
1140 {
1142     int i, j;
1143     attrib_t *att = NULL;
1144     attrib_val_t *eatv, *natv;
1145     attrib_val_t *ev, *nv1, *nv2;
1146     lst_t *values;
1147     boolean_t found;
1149
1150     eatv = eatt->att_value;
1151     natv = natt->att_value;
1152
1153     if (eatv->att_val_type == ATT_VAL_TYPE_NULL &&
1154         natv->att_val_type == ATT_VAL_TYPE_NULL) {
1155
1156         /* NULL - NULL -> EMPTY */
1157         att = attrib_duplicate(eatt);
1158         (void) strcpy(att->att_name, "");
1159
1160     } else if (eatv->att_val_type == ATT_VAL_TYPE_NULL ||
1161                (eatv->att_val_type == ATT_VAL_TYPE_VALUE &&
1162                 natv->att_val_type == ATT_VAL_TYPE_LIST) ||
1163                (eatv->att_val_type == ATT_VAL_TYPE_LIST &&
1164                 natv->att_val_type == ATT_VAL_TYPE_VALUE)) {
1165
1166         /* NULL - X -> ERR, VALUE - LIST -> ERR, LIST - VALUE -> ERR */
1167         util_add_errmsg(errlst, gettext(
1168             "Can not remove attribute \"%s\""),
1169             eatt->att_name));
1170
1171     } else if (natv->att_val_type == ATT_VAL_TYPE_NULL) {
1172
1173         /* X - NULL -> X */
1174         att = attrib_duplicate(eatt);
1175
1176     } else if (eatv->att_val_type == ATT_VAL_TYPE_VALUE &&
1177                natv->att_val_type == ATT_VAL_TYPE_VALUE) {
1178
1179         /* VALUE - VALUE -> {EMPTY | ERR} */
1180         if (attrib_val_equal(eatv, natv) == 0) {
1181             att = ATT_ALLOC();
1182             att->att_name = util_safe_strdup("");
1183             att->att_value = ATT_VAL_ALLOC_NULL();
1184         } else {

```

```

1184         util_add_errmsg(errlst, gettext(
1185             "Can not remove attribute \"%s\""),
1186             eatt->att_name);
1187     }
1188
1189 } else if (eatv->att_val_type == ATT_VAL_TYPE_LIST &&
1190    natv->att_val_type == ATT_VAL_TYPE_LIST) {
1191 /* LIST - LIST -> {EMPTY | ERR | LIST} */
1192 if (attrib_val_equal(eatv, natv) == 0) {
1193     att = ATT_ALLOC();
1194     att->att_name = util_safe_strdup("");
1195     att->att_value = ATT_VAL_ALLOC_NULL();
1196     goto out;
1197 }
1198
1199 for (i = 0; i < lst_size(natv->att_val_values); i++) {
1200     nvl = lst_at(natv->att_val_values, i);
1201     for (j = 0; j < lst_size(natv->att_val_values);
1202         j++) {
1203         nv2 = lst_at(natv->att_val_values, j);
1204         if (i != j && attrib_val_equal(nvl, nv2) == 0) {
1205             util_add_errmsg(errlst, gettext(
1206                 "Duplicate values, can not remove"
1207                 " attribute \"%s\""),
1208                 eatt->att_name);
1209             goto out;
1210         }
1211     }
1212
1213     found = B_FALSE;
1214     for (j = 0; j < lst_size(eatv->att_val_values);
1215         j++) {
1216         ev = lst_at(eatv->att_val_values, j);
1217         if (attrib_val_equal(nvl, ev) == 0) {
1218             found = B_TRUE;
1219             break;
1220         }
1221
1222     if (!found) {
1223         util_add_errmsg(errlst, gettext(
1224             "Value not found, can not remove"
1225             " attribute \"%s\""),
1226             eatt->att_name);
1227         goto out;
1228     }
1229 }
1230
1231 values = util_safe_malloc(sizeof (lst_t));
1232 lst_create(values);
1233 for (i = 0; i < lst_size(eatv->att_val_values); i++) {
1234     ev = lst_at(eatv->att_val_values, i);
1235     found = B_FALSE;
1236     for (j = 0; j < lst_size(natv->att_val_values);
1237         j++) {
1238         nvl = lst_at(natv->att_val_values, j);
1239         if (attrib_val_equal(ev, nvl) == 0) {
1240             found = B_TRUE;
1241             break;
1242         }
1243     }
1244
1245     if (!found) {
1246         lst_insert_tail(values,
1247                         attrib_val_duplicate(ev));
1248     }
1249 }
```

```

1250     }
1251     att = ATT_ALLOC();
1252     att->att_name = util_safe_strdup(eatt->att_name);
1253     att->att_value = ATT_VAL_ALLOC_LIST(values);
1254 }
1255
1256 out:
1257     return (att);
1258 }
1259
1260 attrib_t *
1261 attrib_merge(attrib_t *eatt, attrib_t *natt, int flags, lst_t *errlst)
1262 {
1263     attrib_t *att = NULL;
1264
1265     VERIFY(SEQU(eatt->att_name, natt->att_name));
1266
1267     if (flags & F_MOD_ADD) {
1268         att = attrib_merge_add(eatt, natt);
1269     } else if (flags & F_MOD_REMOVE) {
1270         att = attrib_merge_remove(eatt, natt, errlst);
1271     }
1272
1273     return (att);
1274 }
1275
1276 void
1277 attrib_merge_attrib_lst(lst_t **eattrs, lst_t *nattrs, int flags,
1278     lst_t *errlst)
1279 {
1280     lst_t * attrs = NULL;
1281     int i, j;
1282     attrib_t *att, *natt, *eatt;
1283     boolean_t found;
1284
1285     if (flags & F_MOD_ADD) {
1286         attrs = util_safe_malloc(sizeof (lst_t));
1287         lst_create(attrs);
1288
1289         for (i = 0; i < lst_size(*eattrs); i++) {
1290             eatt = lst_at(*eattrs, i);
1291             found = B_FALSE;
1292             for (j = 0; j < lst_size(nattrs); j++) {
1293                 natt = lst_at(nattrs, j);
1294                 if (SEQU(eatt->att_name, natt->att_name)) {
1295                     found = B_TRUE;
1296                     break;
1297                 }
1298             }
1299
1300             att = found ? attrib_merge(eatt, natt, flags, errlst) :
1301                         attrib_duplicate(eatt);
1302             if (att == NULL) {
1303                 attrib_free_lst(attrs);
1304                 UTIL_FREE_SNLL(attrs);
1305                 goto out;
1306             }
1307             lst_insert_tail(attrs, att);
1308         }
1309     }
1310
1311     for (i = 0; i < lst_size(nattrs); i++) {
1312         natt = lst_at(nattrs, i);
1313         found = B_FALSE;
1314         for (j = 0; j < lst_size(*eattrs); j++) {
1315             eatt = lst_at(*eattrs, j);
1316
1317             if (SEQU(natt->att_name, eatt->att_name)) {
1318                 found = B_TRUE;
1319                 break;
1320             }
1321
1322             att = found ? attrib_merge(natt, eatt, flags, errlst) :
1323                         attrib_duplicate(natt);
1324             if (att == NULL) {
1325                 attrib_free_lst(attrs);
1326                 UTIL_FREE_SNLL(attrs);
1327                 goto out;
1328             }
1329             lst_insert_tail(attrs, att);
1330         }
1331     }
1332
1333     if (errlst != NULL) {
1334         attrib_free_lst(attrs);
1335         UTIL_FREE_SNLL(attrs);
1336         UTIL_FREE_SNLL(errlst);
1337     }
1338
1339     return (attrs);
1340 }
1341
1342 attrib_t *
1343 attrib_merge_attrib_list(attrib_t *eatts, attrib_t *natts, int flags,
1344     lst_t *errlst)
1345 {
1346     attrib_t *att = NULL;
1347
1348     if (errlst != NULL) {
1349         attrib_free_lst(eatts);
1350         UTIL_FREE_SNLL(eatts);
1351         attrib_free_lst(natts);
1352         UTIL_FREE_SNLL(natts);
1353     }
1354
1355     return (att);
1356 }
```

```

1316             if (SEQU(natt->att_name, eatt->att_name)) {
1317                 found = B_TRUE;
1318                 break;
1319             }
1320         if (found)
1321             continue;
1322     lst_insert_tail(attrs, attrib_duplicate(natt));
1323 }
1326 } else if (flags & (F_MOD_Rem | F_MOD_Sub)) {
1328     for (i = 0; i < lst_size(natrs); i++) {
1329         natt = lst_at(natrs, i);
1330         for (j = 0; j < lst_size(natrs); j++) {
1331             att = lst_at(natrs, j);
1332             if (SEQU(natt->att_name, att->att_name) &&
1333                 i != j) {
1334                 util_add_errmsg(errlst, gettext(
1335                     "Duplicate Attributes \"%s\"", natt->att_name));
1336                 goto out;
1337             }
1338         }
1339     }
1340     found = B_FALSE;
1341     for (j = 0; j < lst_size(*eattr); j++) {
1342         eatt = lst_at(*eattr, j);
1343         if (SEQU(eatt->att_name, natt->att_name)) {
1344             found = B_TRUE;
1345             break;
1346         }
1347     }
1348     if (!found) {
1349         util_add_errmsg(errlst, gettext(
1350             "Project does not contain \"%s\"", natt->att_name));
1351         goto out;
1352     }
1353 }
1354
1355 attrs = util_safe_malloc(sizeof (lst_t));
1356 lst_create(attrs);
1358
1360 for (i = 0; i < lst_size(*eattr); i++) {
1361     eatt = lst_at(*eattr, i);
1362     found = B_FALSE;
1363     for (j = 0; j < lst_size(natrs); j++) {
1364         natt = lst_at(natrs, j);
1365         if (SEQU(eatt->att_name, natt->att_name)) {
1366             found = B_TRUE;
1367             break;
1368         }
1369     }
1370     if (flags & F_MOD_Rem) {
1371         att = found ?
1372             attrib_merge(eatt, natt, flags, errlst) :
1373             attrib_duplicate(eatt);
1374     } else if (flags & F_MOD_Sub) {
1375         att = attrib_duplicate(found ? natt : eatt);
1376     }
1377
1378     if (att == NULL) {
1379         attrib_free_lst(attrs);
1380         UTIL_FREE_SNULL(attrs);
1381     }

```

```

1382             goto out;
1383         } else if (SEQU(att->att_name, "")) {
1384             attrib_free(att);
1385         } else {
1386             lst_insert_tail(attrs, att);
1387         }
1388     }
1389 } else if (flags & F_MOD_REP) {
1390     attrs = util_safe_malloc(sizeof (lst_t));
1391     lst_create(attrs);
1392     for (i = 0; i < lst_size(natrs); i++) {
1393         natt = lst_at(natrs, i);
1394         lst_insert_tail(attrs, attrib_duplicate(natt));
1395     }
1396 }
1397
1398 out:
1399     if (attrs != NULL) {
1400         attrib_free_lst(*eattr);
1401         free(*eattr);
1402         *eattr = attrs;
1403     }
1404 }

```

new/usr/src/cmd/projadd/common/attrib.h

1

622 Thu Dec 15 19:51:09 2016

new/usr/src/cmd/projadd/common/attrib.h

Want projadd, projdel and projmod in C.

```
1 #ifndef _PROJENT_ATTRIB_H  
2 #define _PROJENT_ATTRIB_H
```

```
5 #include <sys/types.h>  
6 #include <regex.h>  
7 #include <project.h>  
8 #include <sys/varargs.h>
```

```
10 #include "projent.h"  
11 #include "lst.h"
```

```
13 #ifdef __cplusplus  
14 extern "C" {  
15 #endif
```

```
17 extern char *attrib_lst_tostring(lst_t *);  
18 extern lst_t *attrib_parse_attributes(char *, int, lst_t *);  
19 extern void attrib_free_lst(lst_t *);  
20 extern char *attrib_tostring(void *);  
21 extern void attrib_sort_lst(lst_t *);  
22 extern int attrib_validate_lst(lst_t *, lst_t *);  
23 extern void attrib_merge_attrib_lst(lst_t **, lst_t *, int, lst_t *);
```

```
25 #ifdef __cplusplus  
26 }  
27 #endif  
28 #endif /* _PROJENT_ATTRIB_H */
```

```
new/usr/src/cmd/projadd/common/lst.c
```

```
*****
1728 Thu Dec 15 19:51:09 2016
new/usr/src/cmd/projadd/common/lst.c
Want projadd, projdel and projmod in C.
*****
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <libintl.h>
4 #include "lst.h"
5 #include "util.h"

8 void
9 lst_create(lst_t *plst)
10 {
11     plst->csz = 0;
12     plst->tsz = 0;
13     plst->buf = NULL;
14 }

16 int
17 lst_is_empty(lst_t *plst)
18 {
19     return (plst->csz == 0);
20 }

22 void
23 lst_insert_head(lst_t *plst, void *ndata)
24 {
25     int i;
26     if (plst->csz == plst->tsz) {
27         plst->tsz = (plst->tsz == 0) ? 1 : plst->tsz * 2;
28         plst->buf = util_safe_realloc(plst->buf,
29             plst->tsz * sizeof (void *));
30     }

32     for (i = plst->csz; i > 0; i--)
33         plst->buf[i] = plst->buf[i - 1];

35     plst->buf[0] = ndata;
36     plst->csz++;
37 }

40 void
41 lst_insert_tail(lst_t *plst, void *ndata)
42 {
43     if (plst->csz == plst->tsz) {
44         plst->tsz = (plst->tsz == 0) ? 1 : plst->tsz * 2;
45         plst->buf = util_safe_realloc(plst->buf,
46             plst->tsz * sizeof (void *));
47     }

49     plst->buf[plst->csz++] = ndata;
50 }

52 int
53 lst_remove(lst_t *plst, void *rdata)
54 {
55     int i, idx = -1;
56     for (i = 0; i < plst->csz; i++) {
57         if (plst->buf[i] == rdata) {
58             idx = i;
59             break;
60         }
61     }
62 }
```

```
1
```

```
new/usr/src/cmd/projadd/common/lst.c
*****
62     if (idx >= 0) {
63         for (i = idx; i < plst->csz - 1; i++)
64             plst->buf[i] = plst->buf[i + 1];
65
66         if (--plst->csz == 0) {
67             plst->tsz = 0;
68             free(plst->buf);
69             plst->buf = NULL;
70         }
71         return (0);
72     }
73     return (-1);
74 }

76 void *
77 lst_at(lst_t *plst, int idx)
78 {
79     if (idx < 0 || idx >= plst->csz) {
80         (void) fprintf(stderr, gettext(
81             "error accessing element outside lst\n"));
82         exit(1);
83     }
84     return (plst->buf[idx]);
85 }

87 void *
88 lst_replace_at(lst_t *plst, int idx, void *ndata)
89 {
90     void *odata;
91
92     if (idx < 0 || idx >= plst->csz) {
93         (void) fprintf(stderr, gettext(
94             "error accessing element outside lst\n"));
95         exit(1);
96     }
97     odata = plst->buf[idx];
98     plst->buf[idx] = ndata;
99     return (odata);
100 }

102 int
103 lst_size(lst_t *plst)
104 {
105     return (plst->csz);
106 }
```

```
2
```

```
new/usr/src/cmd/projadd/common/lst.h
```

```
1
```

```
*****
509 Thu Dec 15 19:51:09 2016
new/usr/src/cmd/projadd/common/lst.h
Want projadd, projdel and projmod in C.
*****
1 #ifndef _PROJENT_LST_H
2 #define _PROJENT_LST_H

5 /*
6  * #include <stdlib.h>
7 */

9 #ifdef __cplusplus
10 extern "C" {
11 #endif

15 typedef struct lst_s {
16     int csz;
17     int tsz;
18     void **buf;
19 } lst_t;

22 void lst_create(lst_t *);
23 int lst_is_empty(lst_t *);
24 void lst_insert_head(lst_t *, void *);
25 void lst_insert_tail(lst_t *, void *);

27 int lst_remove(lst_t *, void *);
28 void *lst_at(lst_t *, int);
29 void *lst_replace_at(lst_t *, int, void *);
30 int lst_size(lst_t *);

34 #ifdef __cplusplus
35 }
36#endif
37#endif /* _PROJENT_LST_H */
```

new/usr/src/cmd/projadd/common/projent.c

```
*****
18404 Thu Dec 15 19:51:09 2016
new/usr/src/cmd/projadd/common/projent.c
Want projadd, projdel and projmod in C.
*****
1 #include <sys/types.h>
2 #include <sys/stat.h>
3 #include <fcntl.h>
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <string.h>
7 #include <errno.h>
8 #include <locale.h>
9 #include <stddef.h>
10 #include <limits.h>
11 #include <pwd.h>
12 #include <grp.h>
13 #include <unistd.h>
14 #include <rctl.h>
15 #include <regex.h>
16 #include <ctype.h>

18 #include "projent.h"
19 #include "attrib.h"
20 #include "util.h"

23 #define BOSTR_REG_EXP   "^\n"
24 #define EOSTR_REG_EXP  "$\n"
25 #define IDENT_REG_EXP  "[[:alpha:]][[:alnum:]_.-]*"
26 #define PRJID_REG_EXP  "[[:digit:]]+"
27 #define USERN_REG_EXP  "!?[:alpha:]][[:alnum:]_.-]*"
28 #define GRUPN_REG_EXP  "!?[:alnum:]][[:alnum:]]*"

30 #define TO_EXP(X)      BOSTR_REG_EXP X EOSTR_REG_EXP

32 #define PROJN_EXP      TO_EXP(IDENT_REG_EXP)
33 #define PRJID_EXP     TO_EXP(PRJID_REG_EXP)
34 #define USERN_EXP     TO_EXP(USERN_REG_EXP)
35 #define GRUPN_EXP     TO_EXP(GRUPN_REG_EXP)

37 /*ARGSUSED*/
38 int
39 projent_validate_name(char *pname, lst_t *errlst)
40 {
41     /* Do nothing, as any parse-able project name is valid */
42     return (0);
43 }

45 /*ARGSUSED*/
46 int
47 projent_validate_comment(char *comment, lst_t *errlst)
48 {
49     /* Do nothing, as any parse-able project name is valid */
50     return (0);
51 }

53 int
54 projent_validate_users(char *users, lst_t *errlst)
55 {
56     char *susrs, *usrs, *usr;
57     char *u, **ulast, **ulist;
58     int ret = 0;
59     int i;

61     susrs = usrs = util_safe_strdup(users);
```

1

new/usr/src/cmd/projadd/common/projent.c

```
62     ulast = ulist = util_safe_zmalloc(
63         (strlen(users) + 1) * sizeof (char *));
64     while ((usr = strsep(&susrs, ",")) != NULL) {
65         if (*usr == '!')
66             usr++;
67         if ((*usr == '\0') || (strcmp(usr, "") == 0))
68             continue;
69
70         if (getpwnam(usr) == NULL) {
71             util_add_errmsg(errlst, gettext(
72                 "User \"%s\" does not exist"), usr);
73             ret = 1;
74         }
75         for (i = 0; (u = ulist[i]) != NULL; i++) {
76             if (strcmp(u, usr) == 0) {
77                 util_add_errmsg(errlst, gettext(
78                     "Duplicate user names \"%s\""), usr);
79                 ret = 1;
80             }
81         }
82         /* Add the user to the temporary list if not found */
83         if (u == NULL) {
84             *ulast++ = usr;
85         }
86     }
87     free(ulist);
88     free(susrs);
89     return (ret);
90 }

92 int
93 projent_validate_groups(char *groups, lst_t *errlst)
94 {
95     char *sgrps, *grps, *grp;
96     char *g, **glast, **glist;
97     int ret = 0;
98     int i;

100    sgrps = grps = util_safe_strdup(groups);
101    glast = glist = util_safe_zmalloc(
102        (strlen(groups) + 1) * sizeof (char *));
103    while ((grp = strsep(&sgrps, ",")) != NULL) {
104        if (*grp == '!')
105            grp++;
106        if ((*grp == '\0') || (strcmp(grp, "") == 0))
107            continue;
108
109        if (getgrnam(grp) == NULL) {
110            util_add_errmsg(errlst, gettext(
111                "Group \"%s\" does not exist"), grp);
112            ret = 1;
113        }
114        for (i = 0; (g = glist[i]) != NULL; i++) {
115            if (strcmp(g, grp) == 0) {
116                util_add_errmsg(errlst, gettext(
117                    "Duplicate group names \"%s\""), grp);
118                ret = 1;
119            }
120        }
121        /* Add the group to the temporary list if not found */
122        if (g == NULL) {
123            *glast++ = grp;
124        }
125    }
126    free(glist);
127    free(sgrps);
```

2

```

128     return (ret);
129 }

131 int
132 projent_validate_attributes(lst_t *attrs, lst_t *errlst)
133 {
134     return (attrib_validate_lst(attrs, errlst));
135 }

137 char *
138 projent_tostring(projent_t *ent)
139 {
140     char *attrs;
141     char *ret = NULL;
142     if ((attrs = attrib_lst_tostring(ent->attrs)) != NULL) {
143         (void) asprintf(&ret, "%s:%ld:%s:%s:%s",
144             ent->projname,
145             ent->projid,
146             ent->comment,
147             ent->userlist,
148             ent->grouplist,
149             attrs);
150         free(attrs);
151     }
152     return (ret);
153 }

155 int
156 projent_validate(projent_t *pent, int flags, lst_t *errlst)
157 {
158     char *str;
159
160     (void) projent_validate_name(pent->projname, errlst);
161     (void) projent_validate_projid(pent->projid, flags, errlst);
162     (void) projent_validate_comment(pent->comment, errlst);
163     (void) projent_validate_users(pent->userlist, errlst);
164     (void) projent_validate_groups(pent->grouplist, errlst);
165     (void) projent_validate_attributes(pent->attrs, errlst);
166
167     if ((str = projent_tostring(pent)) == NULL) {
168         util_add_errmsg(errlst, gettext("error allocating memory"));
169     } else {
170         if (strlen(str) > (PROJECT_BUFSZ - 2)) {
171             util_add_errmsg(errlst, gettext(
172                 "projent line too long"));
173         }
174         free(str);
175     }
176     return (lst_is_empty(errlst) == 0);
177 }

179 int
180 projent_validate_lst(lst_t *plst, int flags, lst_t *errlst)
181 {
182     int e, i, idx;
183     projent_t *ent;
184     char *pnames = NULL;
185     projid_t *pids = NULL;
186     int ret = 0;
187
188     idx = 0;
189     for (e = 0; e < lst_size(plst); e++) {
190         ent = lst_at(plst, e);
191         /* Check for duplicate projname */
192         if (pnames != NULL && strstr(pnames, ent->projname) != NULL) {
193             util_add_errmsg(errlst, gettext(

```

```

194                                         "Duplicate project name %s"), ent->projname));
195                                         ret++;
196 }
197
198     /* Check for duplicate projid if DUP is not allowed */
199     if (!(flags & F_PAR_DUP) && pids != NULL) {
200         for (i = 0; i < idx; i++) {
201             if (ent->projid == pids[i]) {
202                 util_add_errmsg(errlst, gettext(
203                     "Duplicate proid %d"), ent->projid);
204                 ret++;
205                 break;
206             }
207         }
208     }
209
210     /* Add the projname an projid to out temp list */
211     pnames = UTIL_STR_APPEND2(pnames, "|", ent->projname);
212     pids = util_safe_realloc(pids, (idx + 1) * sizeof (projid_t));
213     pids[idx] = ent->projid;
214     idx++;
215
216     /* Validate the projet */
217     ret += projent_validate(ent, flags, errlst);
218 }
219
220     free(pnames);
221     free(pids);
222
223     return (ret);
224 }

226 void
227 projent_free_attributes(lst_t *attribs)
228 {
229     attrib_free_lst(attribs);
230 }

232 void
233 projent_sort_attributes(lst_t *attribs)
234 {
235     attrib_sort_lst(attribs);
236 }

238 char *
239 projent_attrib_tostring(void *attrib)
240 {
241     return (attrib_tostring(attrib));
242 }

244 char *
245 projent_attrib_lst_tostring(lst_t *lst)
246 {
247     return (attrib_lst_tostring(lst));
248 }

250 void
251 projent_merge_attributes(lst_t **eattrs, lst_t *nattrs, int flags,
252     lst_t *errlst)
253 {
254     attrib_merge_attrib_lst(eattrs, nattrs, flags, errlst);
255 }

257 lst_t *
258 projent_parse_attributes(char *attribs, int flags, lst_t *errlst)
259 {

```

new/usr/src/cmd/projadd/common/projent.c

5

```

260         return (attrib_parse_attributes(attribs, flags, errlst));
261     }

263 void
264 projent_merge_usrgrp(char *usrgrp, char **elist, char *nlist,
265     int flags, lst_t *errlst)
266 {
267     char *res = NULL;
268     char *seusrs, *eusers, *eusr;
269     char *snusrs, *nusrs, *nusr;
270     char *snlusrs, *nlusrs, *nlusr;
271     char *sep;
272     int i, j;

274     sep = (flags & F_PAR_SPC) ? " , : ,";
275
276     if (flags & F_MOD_ADD) {
277         res = util_safe_strdup(*elist);

279         snusrs = nusrs = util_safe_strdup(nlist);
280         while ((nusr = strsep(&nusrs, sep)) != NULL) {
281             if (*nusr == '\0')
282                 continue;
283             seusers = eusers = util_safe_strdup(*elist);
284             while ((eusr = strsep(&seusers, sep)) != NULL) {
285                 if (*eusr == '\0')
286                     continue;
287                 if (strcmp(eusr, nusr) == 0) {
288                     util_add_errmsg(errlst, gettext(
289                         "Project already contains"
290                         " \"%s\"", usrgrp, nusr));
291                     UTIL_FREE_SNLL(res);
292                     free(seusers);
293                     free(snuusers);
294                     goto out;
295                 }
296             }
297             free(seusers);
298             /* Append nusr to the result */
299             if (*res != '\0')
300                 res = UTIL_STR_APPEND1(res, ",");
301             res = UTIL_STR_APPEND1(res, nusr);
302         }
303         free(snuusers);
304     } else if (flags & F_MOD_REM) {
305
306         snusrs = nusrs = util_safe_strdup(nlist);
307         for (i = 0; (nusr = strsep(&nusrs, sep)) != NULL; i++) {
308             if (*nusr == '\0')
309                 continue;
310             snlusrs = nlusrs = util_safe_strdup(nlist);
311             for (j = 0; (nlusr = strsep(&nlusrs, sep)) != NULL;
312                  j++) {
313                 if (i != j && strcmp(nusr, nlusr) == 0) {
314                     util_add_errmsg(errlst, gettext(
315                         "Duplicate %s name \"%s\"", usrgrp, nusr));
316                     free(snlusrs);
317                     free(snuusers);
318                     goto out;
319                 }
320             }
321         }
322         free(snlusrs);

324         seusers = eusers = util_safe_strdup(*elist);
325         while ((eusr = strsep(&seusers, sep)) != NULL) {

```

new/usr/src/cmd/projadd/common/projent.c

```

326                     if (strcmp(nusr, eusr) == 0) {
327                         break;
328                     }
329                     free(seusrs);
330
331                     if (eusr == NULL) {
332                         util_add_errmsg(errlst, gettext(
333                             "Project does not contain %s name \\"%s\\"),
334                             usrgrp, nusr);
335                         free(snusrs);
336                         goto out;
337                     }
338                 }
339             }
340             free(snusrs);

343             res = util_safe_zmalloc(1);
344             seusrs = eusr = util_safe_strdup(*elist);
345             while ((eusr = strsep(&eusr, sep)) != NULL) {
346                 if (*eusr == '\0')
347                     continue;
348                 snusrs = nusr = util_safe_strdup(nlist);
349                 while ((nusr = strsep(&nusr, sep)) != NULL) {
350                     if (strcmp(eusr, nusr) == 0) {
351                         break;
352                     }
353                 }
354                 free(snusrs);

356                 if (nusr == NULL) {
357                     if (*res != '\0')
358                         res = UTIL_STR_APPEND1(res, ",");
359                     res = UTIL_STR_APPEND1(res, eusr);
360                 }
361             }
362             free(seusrs);
363         } else if (flags & F_MOD_SUB || flags & F_MOD_REP) {
364             res = util_safe_strdup(nlist);
365         }
366     }
367
368     out:
369     if (res != NULL) {
370         free(*elist);
371         *elist = res;
372     }
373 }

374 char *
375 projent_parse_users(char *nlist, int flags, lst_t *errlst)
376 {
377     char *ulist = NULL;
378     char *susrs, *usr, *usr;
379     regex_t usernexp;
380     char *sep;

382     if (regcomp(&usernexp, USERN_EXP, REG_EXTENDED) != 0) {
383         util_add_errmsg(errlst, gettext(
384             "Failed to compile regular expression: \\"%s\\"),
385             USERN_EXP);
386         goto out;
387     }

389     sep = (flags & F_PAR_SPC) ? " , : ",";
390     susrs = usrs = util_safe_strdup(nlist);
391     ulist = util_safe_zmalloc(1);

```

```

393     while ((usr = strsep(&susrs, sep)) != NULL) {
394         if (*usr == '\0')
395             continue;
396
397         if (regexec(&usernexp, usr, 0, NULL, 0) != 0 &&
398             strcmp(usr, "") != 0 &&
399             strcmp(usr, "!*") != 0) {
400             util_add_errmsg(errlst, gettext(
401                 "Invalid user name \"%s\"", usr));
402             UTIL_FREE_SNLL(ulist);
403             break;
404         }
405         /* Append ',' first if required */
406         if (*ulist != '\0')
407             ulist = UTIL_STR_APPEND1(ulist, ",");
408         ulist = UTIL_STR_APPEND1(ulist, usr);
409     }
410
411     free(susrs);
412     regfree(&usernexp);
413 out:
414     return (ulist);
415 }
416
417 char *
418 projent_parse_groups(char *nlist, int flags, lst_t *errlst)
419 {
420     char *glist = NULL;
421     char *sgrps, *grps, *grp;
422     regex_t groupnexp;
423     char *sep;
424
425     if (regcomp(&groupnexp, GRUPN_EXP, REG_EXTENDED) != 0) {
426         util_add_errmsg(errlst, gettext(
427             "Failed to compile regular expression: \"%s\"", GRUPN_EXP));
428         goto out;
429     }
430
431     sep = (flags & F_PAR_SPC) ? " ,": ",";
432     sgrps = grps = util_safe_strdup(nlist);
433     glist = util_safe_zmalloc(1);
434
435     while ((grp = strsep(&grps, sep)) != NULL) {
436         if (*grp == '\0')
437             continue;
438
439         if (regexec(&groupnexp, grp, 0, NULL, 0) != 0 &&
440             strcmp(grp, "") != 0 &&
441             strcmp(grp, "!*") != 0) {
442             util_add_errmsg(errlst, gettext(
443                 "Invalid group name \"%s\"", grp));
444             UTIL_FREE_SNLL(glist);
445             break;
446         }
447         /* Append ',' first if required */
448         if (*glist != '\0')
449             glist = UTIL_STR_APPEND1(glist, ",");
450         glist = UTIL_STR_APPEND1(glist, grp);
451
452     }
453
454     free(sgrps);
455     regfree(&groupnexp);
456 out:
457 
```

```

458         return (glist);
459     }
460
461     int
462     projent_parse_comment(char *comment, lst_t *errlst)
463     {
464         int ret = 0;
465         if (strchr(comment, ':') != NULL) {
466             util_add_errmsg(errlst, gettext(
467                 "Invalid Comment \"%s\": should not contain ':'"),
468                 comment);
469             ret = 1;
470         }
471     }
472     return (ret);
473
474     int
475     projent_validate_unique_id(lst_t *plst, projid_t projid, lst_t *errlst)
476     {
477         int e;
478         projent_t *ent;
479         for (e = 0; e < lst_size(plst); e++) {
480             ent = lst_at(plst, e);
481             if (ent->projid == projid) {
482                 util_add_errmsg(errlst, gettext(
483                     "Duplicate projid \"%d\"", projid));
484                 return (1);
485             }
486         }
487     }
488     return (0);
489
490     int
491     projent_validate_projid(projid_t projid, int flags, lst_t *errlst)
492     {
493         projid_t maxprojid;
494         maxprojid = (flags & F_PAR_RES) ? 0 : 100;
495
496         if (projid < maxprojid) {
497             util_add_errmsg(errlst, gettext(
498                 "Invalid projid \"%d\": "
499                 "must be >= 100"),
500                 projid);
501             return (1);
502         }
503     }
504     return (0);
505
506     int
507     projent_parse_projid(char *projidstr, projid_t *pprojid, lst_t *errlst)
508     {
509         char *ptr;
510         long long llid;
511         regex_t prjidexp;
512         int ret = 0;
513
514         if (regcomp(&prjidexp, PRJID_EXP, REG_EXTENDED) != 0) {
515             util_add_errmsg(errlst, gettext(
516                 "Failed to compile regular expression: \"%s\"", PRJID_EXP));
517             return (1);
518         }
519
520         if (regexec(&prjidexp, projidstr, 0, NULL, 0) != 0) {
521             util_add_errmsg(errlst, gettext("Invalid project id: \"%s\"", projidstr));
522         }
523     }
524 
```

```

524         ret = 1;
525         goto out;
526     }
528
529     llid = strtoll(projidstr, &ptr, 10);
530
531     /* projid should be a positive number */
532     if (llid == 0 && errno == ERANGE && *ptr != '\0') {
533         util_add_errmsg(errlst, gettext("Invalid project id: \"%s\""),
534                         projidstr);
535         ret = 1;
536         goto out;
537     }
538
539     /* projid should be a positive number >= 0 */
540     if (llid < 0) {
541         util_add_errmsg(errlst, gettext(
542             "Invalid projid \"%lld\": must be >= 0"), llid);
543         ret = 1;
544         goto out;
545     }
546
547     /* projid should be less than UID_MAX */
548     if (llid > INT_MAX) {
549         util_add_errmsg(errlst, gettext(
550             "Invalid projid \"%lld\": must be <= %d"),
551             llid, INT_MAX);
552         ret = 1;
553         goto out;
554     }
555
556     if (pprojid != NULL)
557         *pprojid = llid;
558
559 out:
560     regfree(&prjidexp);
561     return (ret);
562 }
563
564 int
565 projent_validate_unique_name(lst_t *plst, char *pname, lst_t *errlst)
566 {
567     int e;
568     projent_t *ent;
569     for (e = 0; e < lst_size(plst); e++) {
570         ent = lst_at(plst, e);
571         if (strcmp(ent->projname, pname) == 0) {
572             util_add_errmsg(errlst, gettext(
573                 "Duplicate project name \"%s\""), pname);
574             return (1);
575         }
576     }
577     return (0);
578 }
579
580 int
581 projent_parse_name(char *pname, lst_t *errlst)
582 {
583     int ret = 1;
584     regex_t projnexp;
585     if (regcomp(&projnexp, PROJN_EXP, REG_EXTENDED) != 0) {
586         util_add_errmsg(errlst, gettext(
587             "Failed to compile regular expression: \"%s\""),
588             PROJN_EXP);
589         goto out;
590     }

```

```

591         if (regexec(&projnexp, pname, 0, NULL, 0) != 0) {
592             util_add_errmsg(errlst, gettext(
593                 "Invalid project name \"%s\", "
594                 "contains invalid characters"), pname);
595         } else if (strlen(pname) > PROJNAME_MAX) {
596             util_add_errmsg(errlst, gettext(
597                 "Invalid project name \"%s\", "
598                 "name too long"), pname);
599         } else {
600             ret = 0;
601         }
602     }
603     regfree(&projnexp);
604     out:
605     return (ret);
606 }
607 void
608 projent_free(projent_t *ent)
609 {
610     free(ent->projname);
611     free(ent->comment);
612     free(ent->userlist);
613     free(ent->grouplist);
614     attrib_free_lst(ent->attrs);
615     free(ent->attrs);
616 }
617
618 projent_t *
619 projent_parse_components(char *projname, char *idstr, char *comment,
620                         char *users, char *groups, char *attr, int flags, lst_t *errlst)
621 {
622     projent_t *ent;
623     int reterr = 0;
624
625     ent = util_safe_zmalloc(sizeof (projent_t));
626
627     ent->projname = util_safe_strdup(projname);
628     ent->comment = util_safe_strdup(comment);
629
630     reterr += projent_parse_name(ent->projname, errlst);
631     reterr += projent_parse_projid(idstr, &ent->projid, errlst);
632     reterr += projent_parse_comment(ent->comment, errlst);
633     ent->userlist = projent_parse_users(users, flags, errlst);
634     ent->grouplist = projent_parse_groups(groups, flags, errlst);
635     ent->attrs = projent_parse_attributes(attr, flags, errlst);
636
637     if (reterr > 0 || ent->userlist == NULL ||
638         ent->grouplist == NULL || ent->attrs == NULL) {
639         projent_free(ent);
640         UTIL_FREE_SNULL(ent);
641     }
642
643     return (ent);
644 }
645
646 projent_t *
647 projent_parse(char *projstr, int flags, lst_t *errlst)
648 {
649     char *str, *sstr;
650     char *projname, *idstr, *comment, *users, *groups, *attrstr;
651     projent_t *ent;
652
653     if (projstr == NULL)
654         return (NULL);

```

```

656     projname = idstr = comment = users = groups = attrstr = NULL;
657     ent = NULL;
658
659     sstr = str = util_safe_strdup(projstr);
660
661     if ((projname = util_safe_strdup(strsep(&str, ":"))) == NULL ||
662         (idstr = util_safe_strdup(strsep(&str, ":"))) == NULL ||
663         (comment = util_safe_strdup(strsep(&str, ":"))) == NULL ||
664         (users = util_safe_strdup(strsep(&str, ":"))) == NULL ||
665         (groups = util_safe_strdup(strsep(&str, ":"))) == NULL ||
666         (attrstr = util_safe_strdup(strsep(&str, ":"))) == NULL ||
667         strsep(str, ":") != NULL) {
668         util_add_errmsg(errlst, gettext(
669             "Incorrect number of fields. Should have 5 \":\"'s."));
670         goto out;
671     }
672
673     ent = projent_parse_components(projname, idstr, comment, users, groups,
674                                     attrstr, flags, errlst);
675 out:
676     free(sstr);
677     free(projname); free(idstr); free(comment);
678     free(users); free(groups); free(attrstr);
679     return (ent);
680 }
681
682 void projent_free_lst(lst_t *plst)
683 {
684     projent_t *ent;
685
686     if (plst == NULL)
687         return;
688
689     while (!lst_is_empty(plst)) {
690         ent = lst_at(plst, 0);
691         (void) lst_remove(plst, ent);
692         projent_free(ent);
693         free(ent);
694     }
695 }
696
697 }
698
700 void
701 projent_put_lst(char *projfile, lst_t *plst, lst_t *errlst)
702 {
703     char *tmpprojfile, *attrs;
704     FILE *fp;
705     projent_t *ent;
706     struct stat statbuf;
707     int e, ret;
708
709     tmpprojfile = NULL;
710     if (asprintf(&tmpprojfile, "%s.%ld_tmp", projfile, getpid()) == -1) {
711         util_add_errmsg(errlst, gettext(
712             "Failed to allocate memory"));
713         goto out;
714     }
715
716     if (stat(projfile, &statbuf) != 0) {
717         util_add_errmsg(errlst, gettext(
718             "Failed to access %s: %s"),
719             projfile, strerror(errno));
720         goto out;
721     }

```

```

722     if ((fp = fopen(tmpprojfile, "wx")) == NULL) {
723         util_add_errmsg(errlst, gettext(
724             "Cannot create %s: %s"),
725             tmpprojfile, strerror(errno));
726         goto out;
727     }
728
729     for (e = 0; e < lst_size(plst); e++) {
730         ent = lst_at(plst, e);
731         attrs = attrib_lst_tostring(ent->attrs);
732         ret = fprintf(fp, "%s:%ld:%s:%s\n", ent->projname,
733             ent->projid, ent->comment, ent->userlist, ent->grouplist,
734             attrs);
735         free(attrs);
736         if (ret < 0) {
737             util_add_errmsg(errlst, gettext(
738                 "Failed to write to %s: %s"),
739                 tmpprojfile, strerror(errno));
740             /* Remove the temporary file and exit */
741             (void) unlink(tmpprojfile);
742             goto out1;
743         }
744     }
745
746     if (chown(tmpprojfile, statbuf.st_uid, statbuf.st_gid) != 0) {
747         util_add_errmsg(errlst, gettext(
748             "Cannot set ownership of %s: %s"),
749             tmpprojfile, strerror(errno));
750         (void) unlink(tmpprojfile);
751         goto out1;
752     }
753
754     if (rename(tmpprojfile, projfile) != 0) {
755         util_add_errmsg(errlst, gettext(
756             "Cannot rename %s to %s : %s"),
757             tmpprojfile, projfile, strerror(errno));
758         (void) unlink(tmpprojfile);
759         goto out1;
760     }
761
762 out1:
763     (void) fclose(fp);
764 out:
765     free(tmpprojfile);
766 }
767
768 lst_t *
769 projent_get_lst(char *projfile, int flags, lst_t *errlst)
770 {
771     FILE *fp;
772     lst_t *plst;
773     int line = 0;
774     char *buf = NULL, *nlp;
775     size_t cap = 0;
776     projent_t *ent;
777
778     plst = util_safe_malloc(sizeof (lst_t));
779     lst_create(plst);
780
781     if ((fp = fopen(projfile, "r")) == NULL) {
782         if (errno == ENOENT) {
783             /*
784             * There is no project file,
785             * return an empty lst
786             */
787         }
788     }
789
790     return (plst);
791 }

```

```
788         } else {
789             /* Report the error unable to open the file */
790             util_add_errmsg(errlst, gettext(
791                 "Cannot open %s: %s"),
792                 projfile, strerror(errno));
793
794             free(plst);
795             return (NULL);
796         }
797     }
798
799     while ((getline(&buf, &cap, fp)) != -1 && ++line) {
800
801         if ((nlp = strchr(buf, '\n')) != NULL)
802             *nlp = '\0';
803
804         if ((ent = projent_parse(buf, flags, errlst)) != NULL) {
805             lst_insert_tail(plst, ent);
806         } else {
807             /* Report the error */
808             util_add_errmsg(errlst, gettext(
809                 "Error parsing: %s line: %d: \"%s\""),
810                 projfile, line, buf);
811
812             /* free the allocated resources */
813             projent_free_lst(plst);
814             UTIL_FREE_SNULL(plst);
815             goto out;
816         }
817     }
818
819     if (flags & F_PAR_VLD && plst != NULL) {
820         if (projent_validate_lst(plst, flags, errlst) != 0) {
821             projent_free_lst(plst);
822             UTIL_FREE_SNULL(plst);
823         }
824     }
825
826 out:
827     (void) fclose(fp);
828     free(buf);
829     return (plst);
830 }
```

new/usr/src/cmd/projadd/common/projent.h

1

```
*****  
2070 Thu Dec 15 19:51:09 2016  
new/usr/src/cmd/projadd/common/projent.h  
Want projadd, projdel and projmod in C.  
*****  
1 #ifndef _PROJENT_PROJENT_H  
2 #define _PROJENT_PROJENT_H  
  
4 #include <sys/types.h>  
5 #include <regex.h>  
6 #include <project.h>  
7 #include <sys/varargs.h>  
  
9 #include "lst.h"  
  
11 #define F_PAR_VLD      0x0001 /* Run validation after parsing */  
12 #define F_PAR_SPC      0x0002 /* Allow spaces between names */  
13 #define F_PAR_UNT      0x0004 /* Allow units in attribs values */  
14 #define F_PAR_RES      0x0008 /* Allow projid < 100 */  
15 #define F_PAR_DUP      0x0010 /* Allow duplicate projids */  
  
17 #define F_MOD_ADD      0x0100  
18 #define F_MOD_Rem      0x0200  
19 #define F_MOD_Sub      0x0400  
20 #define F_MOD_Rep      0x0800  
  
22 #ifdef __cplusplus  
23 extern "C" {  
24 #endif  
  
27 typedef struct projent {  
28     char *projname;  
29     projid_t projid;  
30     char *comment;  
31     char *userlist;  
32     char *grouplist;  
33     lst_t *atrrs;  
34 } projent_t;  
  
37 extern void projent_free(projent_t *);  
38 extern projent_t *projent_parse(char *, int, lst_t *);  
39 extern projent_t *projent_parse_components(char *, char *, char *, char *,  
40     char *, char *, int, lst_t *);  
41 extern int projent_validate(projent_t *, int, lst_t *);  
42 extern lst_t *projent_get_lst(char *, int, lst_t *);  
43 extern void projent_free_lst(lst_t *);  
44 extern int projent_parse_name(char *, lst_t *);  
45 extern int projent_validate_unique_name(lst_t *, char *, lst_t *);  
46 extern int projent_parse_projid(char *, projid_t *, lst_t *);  
47 extern int projent_validate_projid(projid_t, int, lst_t *);  
48 extern int projent_validate_unique_id(lst_t *, projid_t, lst_t *);  
49 extern int projent_parse_comment(char *, lst_t *);  
50 extern void projent_merge_usrgrp(char *, char **, char *, int, lst_t *);  
51 extern char *projent_parse_users(char *, int, lst_t *);  
52 extern char *projent_parse_groups(char *, int, lst_t *);  
53 extern void projent_merge_attributes(lst_t **, lst_t *, int, lst_t *);  
54 extern lst_t *projent_parse_attributes(char *, int, lst_t *);  
55 extern void projent_sort_attributes(lst_t *);  
56 extern void projent_free_attributes(lst_t *);  
57 extern char *projent_attrib_tostring(void *);  
58 extern char *projent_attrib_lst_tostring(lst_t *);  
59 extern void projent_put_lst(char *, lst_t *, lst_t *);  
  
61 #ifdef __cplusplus
```

new/usr/src/cmd/projadd/common/projent.h

2

```
62 }  
63 #endif  
65 #endif /* _PROJENT_PROJENT_H */
```

new/usr/src/cmd/projadd/common/resctl.c

```
*****
3682 Thu Dec 15 19:51:09 2016
new/usr/src/cmd/projadd/common/resctl.c
Want projadd, projdel and projmod in C.
*****
1 #include <sys/types.h>
2 #include <sys/stat.h>
3 #include <fcntl.h>
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <string.h>
7 #include <errno.h>
8 #include <locale.h>
9 #include <stddef.h>
10 #include <limits.h>
11 #include <rctl.h>
12 #include <regex.h>
13 #include <cctype.h>
14 #include <zone.h>
15 #include <pool.h>
16 #include <sys/pool_impl.h>
17 #include <unistd.h>
18 #include <stropts.h>
20 #include "util.h"
21 #include "resctl.h"

23 sig_t sigs[SIGS_CNT] = {
24     /* Signal names */
25     {"ABRT", RESCTL_SIG_ABRT},
26     {"XRES", RESCTL_SIG_XRES},
27     {"HUP", RESCTL_SIG_HUP},
28     {"STOP", RESCTL_SIG_STOP},
29     {"TERM", RESCTL_SIG_TERM},
30     {"KILL", RESCTL_SIG_KILL},
31     {"XFSZ", RESCTL_SIG_XFSZ},
32     {"XCPU", RESCTL_SIG_XCPU},
33
34     /* Singnal numbers */
35     {"6", RESCTL_SIG_ABRT},
36     {"38", RESCTL_SIG_XRES},
37     {"1", RESCTL_SIG_HUP},
38     {"23", RESCTL_SIG_STOP},
39     {"15", RESCTL_SIG_TERM},
40     {"9", RESCTL_SIG_KILL},
41     {"31", RESCTL_SIG_XFSZ},
42     {"30", RESCTL_SIG_XCPU},
43 };

45 /*
46 * Check the existance of a resource pool in the system
47 */
48 int
49 resctl_pool_exist(char *name)
50 {
51     pool_conf_t *conf;
52     pool_status_t status;
53     int fd;
54
55     /*
56      * Determine if pools are enabled using /dev/pool, as
57      * libpool may not be present.
58      */
59     if (getzoneid() != GLOBAL_ZONEID ||
60         (fd = open("/dev/pool", O_RDONLY)) < 0) {
61         return (1);
62     }
63
64     if (ioctl(fd, POOL_STATUS, &status) < 0) {
65         (void) close(fd);
66         return (1);
67     }
68
69     (void) close(fd);
70
71     if (status.ps_io_state != 1)
72         return (1);
73
74     /* If pools are enabled, assume libpool is present. */
75     if ((conf = pool_conf_alloc()) == NULL)
76         return (1);
77
78     if (pool_conf_open(conf, pool_dynamic_location(), PO_RDONLY)) {
79         pool_conf_free(conf);
80         return (1);
81     }
82
83     if (pool_get_pool(conf, name) == NULL) {
84         (void) pool_conf_close(conf);
85         pool_conf_free(conf);
86         return (1);
87     }
88
89     (void) pool_conf_close(conf);
90     pool_conf_free(conf);
91     return (0);
92 }

93 int
94 resctl_get_info(char *name, resctl_info_t *pinfo)
95 {
96     rctlblk_t *blk1, *blk2, *tmp;
97     rctl_priv_t priv;
98     int ret = 1;
99
100    blk1 = blk2 = tmp = NULL;
101    blk1 = util_safe_malloc(rctlblk_size());
102    blk2 = util_safe_malloc(rctlblk_size());
103
104    if (getrctl(name, NULL, blk1, RCTL_FIRST) == 0) {
105        priv = rctlblk_get_privilege(blk1);
106        while (priv != RCPRIV_SYSTEM) {
107            tmp = blk2;
108            blk2 = blk1;
109            blk1 = tmp;
110            if (getrctl(name, blk2, blk1, RCTL_NEXT) != 0) {
111                goto out;
112            }
113            priv = rctlblk_get_privilege(blk1);
114        }
115    }
116
117    pinfo->value = rctlblk_get_value(blk1);
118    pinfo->flags = rctlblk_get_global_flags(blk1);
119    ret = 0;
120
121 out:
122     free(blk1);
123     free(blk2);
124
125     return (ret);
126 }
```

1

new/usr/src/cmd/projadd/common/resctl.c

```
62     }
63
64     if (ioctl(fd, POOL_STATUS, &status) < 0) {
65         (void) close(fd);
66         return (1);
67     }
68
69     (void) close(fd);
70
71     if (status.ps_io_state != 1)
72         return (1);
73
74     /* If pools are enabled, assume libpool is present. */
75     if ((conf = pool_conf_alloc()) == NULL)
76         return (1);
77
78     if (pool_conf_open(conf, pool_dynamic_location(), PO_RDONLY)) {
79         pool_conf_free(conf);
80         return (1);
81     }
82
83     if (pool_get_pool(conf, name) == NULL) {
84         (void) pool_conf_close(conf);
85         pool_conf_free(conf);
86         return (1);
87     }
88
89     (void) pool_conf_close(conf);
90     pool_conf_free(conf);
91     return (0);
92 }

93 int
94 resctl_get_info(char *name, resctl_info_t *pinfo)
95 {
96     rctlblk_t *blk1, *blk2, *tmp;
97     rctl_priv_t priv;
98     int ret = 1;
99
100    blk1 = blk2 = tmp = NULL;
101    blk1 = util_safe_malloc(rctlblk_size());
102    blk2 = util_safe_malloc(rctlblk_size());
103
104    if (getrctl(name, NULL, blk1, RCTL_FIRST) == 0) {
105        priv = rctlblk_get_privilege(blk1);
106        while (priv != RCPRIV_SYSTEM) {
107            tmp = blk2;
108            blk2 = blk1;
109            blk1 = tmp;
110            if (getrctl(name, blk2, blk1, RCTL_NEXT) != 0) {
111                goto out;
112            }
113            priv = rctlblk_get_privilege(blk1);
114        }
115    }
116
117    pinfo->value = rctlblk_get_value(blk1);
118    pinfo->flags = rctlblk_get_global_flags(blk1);
119    ret = 0;
120
121 out:
122     free(blk1);
123     free(blk2);
124
125     return (ret);
126 }
```

2

```
128 void
129 resctl_get_rule(resctl_info_t *pinfo, resctlrule_t *prule)
130 {
132     prule->resctl_max = pinfo->value;
133     if (pinfo->flags & RCTL_GLOBAL_BYTES) {
134         prule->resctl_type = RESCTL_TYPE_BYTES;
135     } else if (pinfo->flags & RCTL_GLOBAL_SECONDS) {
136         prule->resctl_type = RESCTL_TYPE_SCNDS;
137     } else if (pinfo->flags & RCTL_GLOBAL_COUNT) {
138         prule->resctl_type = RESCTL_TYPE_COUNT;
139     } else {
140         prule->resctl_type = RESCTL_TYPE_UNKWN;
141     }
143     if (pinfo->flags & RCTL_GLOBAL_NOBASIC) {
144         prule->resctl_privs = RESCTL_PRIV_PRIVE | RESCTL_PRIV_PRIVD;
145     } else {
146         prule->resctl_privs = RESCTL_PRIV_ALLPR;
147     }
149     if (pinfo->flags & RCTL_GLOBAL_DENY_ALWAYS) {
150         prule->resctl_action = RESCTL_ACTN_DENY;
151     } else if (pinfo->flags & RCTL_GLOBAL_DENY_NEVER) {
152         prule->resctl_action = RESCTL_ACTN_NONE;
153     } else {
154         prule->resctl_action = RESCTL_ACTN_NONE | RESCTL_ACTN_DENY;
155     }
157     if (pinfo->flags & RCTL_GLOBAL_SIGNAL_NEVER) {
158         prule->resctl_sigs = 0;
159     } else {
160         prule->resctl_action |= RESCTL_ACTN_SIGN;
161         prule->resctl_sigs = RESCTL_SIG_CMN;
162         if (pinfo->flags & RCTL_GLOBAL_CPU_TIME) {
163             prule->resctl_sigs |= RESCTL_SIG_XCPU;
164         }
165         if (pinfo->flags & RCTL_GLOBAL_FILE_SIZE) {
166             prule->resctl_sigs |= RESCTL_SIG_XFSZ;
167         }
168     }
169 }
```

new/usr/src/cmd/projadd/common/resctl.h

```
*****
1386 Thu Dec 15 19:51:09 2016
new/usr/src/cmd/projadd/common/resctl.h
Want projadd, projdel and projmod in C.
*****
1 #ifndef _PROJENT_RESCTL_H
2 #define _PROJENT_RESCTL_H

5 /*
6  * Put includes here if needed to be.
7 */

9 #ifdef __cplusplus
10 extern "C" {
11 #endif

14 #define RESCTL_TYPE_UNKWN      0x00
15 #define RESCTL_TYPE_BYTES      0x01
16 #define RESCTL_TYPE_SCNDS      0x02
17 #define RESCTL_TYPE_COUNT      0x03

19 #define RESCTL_PRIV_PRIVE     0x01
20 #define RESCTL_PRIV_PRIVD     0x02
21 #define RESCTL_PRIV_BASIC     0x04
22 #define RESCTL_PRIV_ALLPR     0x07

24 #define RESCTL_ACTN_NONE      0x01
25 #define RESCTL_ACTN_DENY      0x02
26 #define RESCTL_ACTN_SIGN      0x04
27 #define RESCTL_ACTN_ALLA      0x07

29 #define RESCTL_SIG_ABRT       0x01
30 #define RESCTL_SIG_XRES      0x02
31 #define RESCTL_SIG_HUP        0x04
32 #define RESCTL_SIG_STOP       0x08
33 #define RESCTL_SIG_TERM       0x10
34 #define RESCTL_SIG_KILL       0x20
35 #define RESCTL_SIG_XFSZ      0x40
36 #define RESCTL_SIG_XCPU       0x80

38 #define RESCTL_SIG_CMN        0x3f
39 #define RESCTL_SIG_ALL        0xff

41 typedef struct sig_s {
42     char sig[6];
43     int mask;
44 } sig_t;

46 #define SIGS_CNT           16
47 extern sig_t sigs[SIGS_CNT];

49 typedef struct resctlrule_s {
50     uint64_t resctl_max;
51     uint8_t  resctl_type;
52     uint8_t  resctl_privs;
53     uint8_t  resctl_action;
54     uint8_t  resctl_sigs;
55 } resctlrule_t;

57 typedef struct resctl_info_s {
58     unsigned long long value;
59     int flags;
60 } resctl_info_t;
```

1

new/usr/src/cmd/projadd/common/resctl.h

```
62 extern int resctl_pool_exist(char *);
63 extern int resctl_get_info(char *, resctl_info_t *);
64 extern void resctl_get_rule(resctl_info_t *, resctlrule_t *);

66 #ifdef __cplusplus
67 }
68 #endif
69 #endif /* _PROJENT_RESCTL_H */
```

2

new/usr/src/cmd/projadd/common/util.c

```
*****
6662 Thu Dec 15 19:51:09 2016
new/usr/src/cmd/projadd/common/util.c
Want projadd, projdel and projmod in C.
*****
1 #include <sys/types.h>
2 #include <sys/stat.h>
3 #include <fcntl.h>
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <string.h>
7 #include <errno.h>
8 #include <locale.h>
9 #include <stddef.h>
10 #include <limits.h>
12 #include <project.h>
14 #include <ctype.h>
16 #include "util.h"

20 #define BOSTR_REG_EXP    "^\n"
21 #define EOSTR_REG_EXP    "$"
22 #define FLTNM_REG_EXP    "[[:digit:]]+([\\.][[:digit:]]+)?"
23 #define MODIF_REG_EXP    "[kmgtpeKMGTPE]?"
24 #define UNIT_REG_EXP     "[bsBS]?"
25 #define TOKEN_REG_EXP    "[[:alnum:]\._/=-+]*"
26 #define VALUE_REG_EXP    FLTNM_REG_EXP MODIF_REG_EXP UNIT_REG_EXP

28 #define TO_EXP(X)          BOSTR_REG_EXP X EOSTR_REG_EXP
29 #define TOKEN_EXP          TO_EXP(TOKEN_REG_EXP)
30 #define VALUE_EXP          TO_EXP(VALUE_REG_EXP)

32 #define BYTES_SCALE        1
33 #define SCNDS_SCALE         2

35 char *
36 util_safe_strdup(char *str)
37 {
38     char *ptr;
39     if (str == NULL)
40         return (NULL);
42     if ((ptr = strdup(str)) == NULL) {
43         (void) fprintf(stderr, gettext("error allocating memory"));
44         exit(1);
45     }
46     return (ptr);
47 }

49 void *
50 util_safe_realloc(void *ptr, size_t sz)
51 {
52     if ((ptr = realloc(ptr, sz)) == NULL) {
53         (void) fprintf(stderr, gettext(
54             "error reallocating %d bytes of memory"), sz);
55         exit(1);
56     }
57     return (ptr);
58 }

61 void *
```

1

new/usr/src/cmd/projadd/common/util.c

```
62 util_safe_malloc(size_t sz)
63 {
64     char *ptr;
65     if ((ptr = malloc(sz)) == NULL) {
66         (void) fprintf(stderr, gettext(
67             "error allocating %d bytes of memory\n"), sz);
68         exit(1);
69     }
70     return (ptr);
71 }

73 void *
74 util_safe_zmalloc(size_t sz)
75 {
76     return (memset(util_safe_malloc(sz), 0, sz));
77 }

79 void
80 util_print_errmsgs(lst_t *errlst)
81 {
82     char *errmsg;
83     while (!lst_is_empty(errlst)) {
84         errmsg = lst_at(errlst, 0);
85         (void) lst_remove(errlst, errmsg);
86         (void) fprintf(stderr, "%s\n", errmsg);
87         free(errmsg);
88     }
89 }

92 void
93 util_add_errmsg(lst_t *errlst, char *format, ...)
94 {
95     va_list args;
96     char *errmsg;
98     va_start(args, format);
99     if (vasprintf(&errmsg, format, args) < 0) {
100         va_end(args);
101         (void) fprintf(stderr, gettext(
102             "error allocating memory\n"));
103         exit(1);
104     }
105     va_end(args);
106     lst_insert_tail(errlst, errmsg);
107 }

109 char *
110 util_str_append(char *str, int nargs, ...)
111 {
112     va_list ap;
113     int i, len;
114     char *s;
116     if (str == NULL)
117         str = util_safe_zmalloc(1);
119     len = strlen(str) + 1;
120     va_start(ap, nargs);
121     for (i = 0; i < nargs; i++) {
122         s = va_arg(ap, char *);
123         len += strlen(s);
124         str = util_safe_realloc(str, len);
125         (void) strcat(str, s);
126     }
127     va_end(ap);
```

2

```

128         return (str);
129     }
130
131     char *
132     util_substr(regex_t *reg, regmatch_t *mat, char *str, int idx)
133     {
134         int mat_len;
135         char *ret;
136
137         if (idx < 0 || idx > reg->re_nsub)
138             return (NULL);
139
140         mat_len = mat[idx].rm_eo - mat[idx].rm_so;
141
142         ret = util_safe_malloc(mat_len + 1);
143         *ret = '\0';
144
145         (void) strlcpy(ret, str + mat[idx].rm_so, mat_len + 1);
146
147     }
148
149     typedef struct {
150         char unit;
151         uint64_t val;
152     } scl;
153
154 #define SCLS    7
155
156     int
157     util_scale(char *unit, int scale, uint64_t *res, lst_t *errlst)
158     {
159         int i;
160         scl *sc;
161         scl bscale[SCLS] = {
162             {'\0', 1ULL},
163             {'k', 1024ULL},
164             {'m', 1048576ULL},
165             {'g', 1073741824ULL},
166             {'t', 1099511627776ULL},
167             {'p', 1125899906842624ULL},
168             {'e', 1152921504606846976ULL},
169         };
170
171         scl oscale[SCLS] = {
172             {'\0', 1ULL},
173             {'k', 1000ULL},
174             {'m', 1000000ULL},
175             {'g', 1000000000ULL},
176             {'t', 10000000000000ULL},
177             {'p', 10000000000000000ULL},
178             {'e', 100000000000000000ULL},
179         };
180
181         sc = (scale == BYTES_SCALE) ? bscale : oscale;
182
183         for (i = 0; i < SCLS; i++) {
184             if (tolower(*unit) == sc[i].unit) {
185                 *res = sc[i].val;
186                 return (0);
187             }
188         }
189
190         util_add_errmsg(errlst, gettext(
191             "Invalid scale: %d"), scale);
192
193     }

```

```

196     int
197     util_val2num(char *value, int scale, lst_t *errlst, char **retnum,
198         char **retmod, char **retunit)
199     {
200         int ret = 1;
201         regex_t valueexp;
202         regmatch_t *mat;
203         int nmatch;
204         char *num, *modifier, *unit;
205         char *ptr;
206
207         uint64_t mul64;
208         long double dnum;
209
210         *retnum = *retmod = *retunit = NULL;
211
212         if (regcomp(&valueexp, VALUE_EXP, REG_EXTENDED) != 0) {
213             util_add_errmsg(errlst, gettext(
214                 "Failed to compile regex: '%s'"), VALUE_EXP);
215             return (1);
216         }
217
218         nmatch = valueexp.re_nsub + 1;
219         mat = util_safe_malloc(nmatch * sizeof (regmatch_t));
220
221         if (regexec(&valueexp, value, nmatch, mat, 0) != 0) {
222             util_add_errmsg(errlst, gettext(
223                 "Invalid value: '%s'"), value);
224             regfree(&valueexp);
225             return (1);
226         }
227         regfree(&valueexp);
228
229         num = util_substr(&valueexp, mat, value, 1);
230         modifier = util_substr(&valueexp, mat, value, 3);
231         unit = util_substr(&valueexp, mat, value, 4);
232
233         if ((num == NULL || modifier == NULL || unit == NULL) ||
234             (strlen(modifier) == 0 && strchr(num, '.') != NULL) ||
235             (util_scale(modifier, scale, &mul64, errlst) != 0) ||
236             (scale == BYTES_SCALE && *unit != '\0' && tolower(*unit) != 'b') ||
237             (scale == SCNDS_SCALE && *unit != '\0' && tolower(*unit) != 's') ||
238             (scale == COUNT_SCALE && *unit != '\0') ||
239             (scale == UNKWN_SCALE && *unit != '\0')) {
240                 util_add_errmsg(errlst, gettext("Error near: \"%s\""),
241                     value);
242                 goto out;
243             }
244
245         dnum = strtold(num, &ptr);
246         if (dnum == 0 &&
247             (errno == EINVAL || errno == ERANGE) &&
248             *ptr != '\0') {
249                 util_add_errmsg(errlst, gettext("Invalid value: \"%s\""),
250                     value);
251                 goto out;
252             }
253
254         if (UINT64_MAX / mul64 <= dnum) {
255             util_add_errmsg(errlst, gettext("Too big value: \"%s\""),
256                     value);
257             goto out;
258         }

```

```

260     if (asprintf(retnum, "%llu",
261         (unsigned long long)(mul64 * dnum)) == -1) {
262         goto out;
263     }
264
265     free(num);
266     *retmod = modifier;
267     *retunit = unit;
268     return (0);
269 out:
270     free(num); free(modifier); free(unit);
271     return (ret);
272 }
273
274 void
275 util_free_tokens(char **tokens)
276 {
277     char *token;
278     while ((token = *tokens++) != NULL) {
279         free(token);
280     }
281 }
282
283 char **
284 util_tokenize(char *values, lst_t *errlist)
285 {
286     char *token, *t;
287     char *v;
288     regex_t tokenexp;
289     char **ctoken, **tokens = NULL;
290
291     if (regcomp(&tokenexp, TOKEN_EXP, REG_EXTENDED | REG_NOSUB) != 0)
292         return (tokens);
293
294     /* Assume each character will be a token + NULL terminating value. */
295     ctoken = tokens = util_safe_malloc(
296         (strlen(values) + 1) * sizeof(char *));
297     token = util_safe_malloc(strlen(values) + 1);
298
299     v = values;
300     *ctoken = NULL;
301     while (v < (values + strlen(values))) {
302
303         /* get the next token */
304         t = token;
305         while ((*t = *v++) != '\0') {
306             if (*t == '(' || *t == ')' || *t == ',' || *v == ',') {
307                 *v == '(' || *v == ')' || *v == ',' ?
308                     *++t = '\0';
309                 break;
310             }
311             t++;
312         }
313
314         if (strcmp(token, "(") != 0 && strcmp(token, ")") != 0 &&
315             strcmp(token, ",") != 0) {
316             if (regexec(&tokenexp, token, 0, NULL, 0) != 0) {
317                 util_add_errmsg(errlist, gettext(
318                     "Invalid Character at or near \"%s\""),
319                     token);
320                 util_free_tokens(tokens);
321                 UTIL_FREE_SNLL(tokens);
322                 goto outl;
323             }
324         }
325         *ctoken++ = util_safe_strdup(token);

```

```

326             *ctoken = NULL;
327         }
328         outl:
329         free(token);
330         regfree(&tokenexp);
331         return (tokens);
332     }
333 }

```

```
*****
1106 Thu Dec 15 19:51:09 2016
new/usr/src/cmd/projadd/common/util.h
Want projadd, projdel and projmod in C.
*****
1 #ifndef _PROJENT_UTIL_H
2 #define _PROJENT_UTIL_H

4 #include <sys/types.h>
5 #include <regex.h>
6 #include <sys/varargs.h>

8 #include "lst.h"

10 #ifdef __cplusplus
11 extern "C" {
12 #endif

14 /* UTIL_STR_APPEND */
15 #define UTIL_STR_APPEND1(S, S1) util_str_append((S), 1, (S1))
16 #define UTIL_STR_APPEND2(S, S1, S2) util_str_append((S), 2, (S1), (S2))

18 /* UTIL_FREE_SNULL */
19 #define UTIL_FREE_SNULL(ptr) { \
20     free(ptr); \
21     ptr = NULL; \
22 }

24 #define UNKWN_SCALE      0
25 #define BYTES_SCALE      1
26 #define SCNDS_SCALE       2
27 #define COUNT_SCALE        3

30 extern char *util_safe_strdup(char *);
31 extern void *util_safe_realloc(void*, size_t);
32 extern void *util_safe_malloc(size_t);
33 extern void *util_safe_zmalloc(size_t);
34 extern char **util_tokenize(char *, lst_t *);
35 extern void util_free_tokens(char **);
36 extern char *util_substr(regex_t *, regmatch_t *, char *, int);
37 extern char *util_str_append(char *, int, ...);
38 extern int util_val2num(char *, int, lst_t *, char **, char **, char **);
39 extern void util_add_errmsg(lst_t *, char *format, ...);
40 extern void util_print_errmsgs(lst_t *);

42 #ifdef __cplusplus
43 }
44 #endif
45 #endif /* _PROJENT_UTIL_H */
```

```

new/usr/src/cmd/projadd/projadd/Makefile
*****
1722 Thu Dec 15 19:51:09 2016
new/usr/src/cmd/projadd/projadd/Makefile
Want projadd, projdel and projmod in C.
*****
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 2009 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24 #

26 PROG = projadd
27 OBJS = projadd.o
28 SRCS =$(OBJS:%.o=%.c) $(COMMON_SRCS)

30 include $(SRC)/cmd/Makefile.cmd
31 include $(SRC)/cmd/projadd/Makefile.projadd

33 LDLIBS += -lpool
34 CFLAGS += $(CCVERBOSE) -I${PROJADDCOMMONDIR}
35 CERRWARN += -_gcc=-Wno-uninitialized
36 CERRWARN += -_gcc=-Wno-switch
37 CERRWARN += -_gcc=-Wno-parentheses

39 CPPFLAGS_sparc += -I$(SRC)/uts/sfmmu
40 CPPFLAGS_sparc += -I$(SRC)/uts/sun4u/sunfire
41 CPPFLAGS += $(CPPFLAGS_${MACH})

43 FILEMODE= 0555

45 lint := LINTFLAGS = -muxs -I${PROJADDCOMMONDIR}

47 .KEEP_STATE:

49 all: $(PROG)

51 install: all $(ROOTPROG)

53 $(PROG): $(OBJS) $(COMMON_OBJS)
54     $(LINK.c) -o $(PROG) $(OBJS) $(COMMON_OBJS) $(LDLIBS)
55     $(POST_PROCESS)

57 %.o : ${PROJADDCOMMONDIR}/%.c
58     $(COMPILE.c) -o $@ $<
59     $(POST_PROCESS_O)

61 clean:

```

```

1
new/usr/src/cmd/projadd/projadd/Makefile
*****
2
62      -$(RM) $(PROG) $(OBJS) $(COMMON_OBJS)
64 lint: lint_SRCS
66 include $(SRC)/cmd/Makefile.targ

```

```
new/usr/src/cmd/projadd/projadd.c
```

```
*****
5012 Thu Dec 15 19:51:09 2016
new/usr/src/cmd/projadd/projadd/projadd.c
Want projadd, projdel and projmod in C.
*****
```

```
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 #include <stdio.h>
23 #include <stdlib.h>
24 #include <libintl.h>
25 #include <locale.h>
26 #include <errno.h>
27 #include <string.h>
28 #include <stddef.h>
29 #include <sys/types.h>
30
31 #include "projent.h"
32 #include "util.h"
33
34 #define CHECK_ERRORS_FREE_PLST(errlst, plist, attrs, ecode) { \
35     if (!lst_is_empty(errlst)) { \
36         util_print_errmsgs(errlst); \
37         if (plist != NULL) { \
38             projent_free_lst(plist); \
39             free(plist); \
40         } \
41         free(attrs); \
42         usage(); \
43         exit(ecode); \
44     } \
45 }
46
47 /*
48 * Print usage
49 */
50 static void
51 usage(void)
52 {
53     (void) fprintf(stderr, gettext(
54         "Usage:\n"
55         "projadd [-n] [-f filename] [-p projid [-o]] [-c comment]\n"
56         "          [-U user[,user...]] [-G group[,group...]]\n"
57         "          [-K name[=value[,value...]]] project\n"));
58 }
59
60 */


```

```
1
```

```
new/usr/src/cmd/projadd/projadd/projadd.c
```

```
62     * main()
63     */
64 int
65 main(int argc, char **argv)
66 {
67     int e, c;
68
69     extern char *optarg;
70     extern int optind, optopt;
71     projid_t maxpjid = 99;
72     lst_t *plist = NULL;
73     int flags = 0;
74     char *projfile = PROJF_PATH;
75
76     projent_t *ent;
77     boolean_t nflag, pflag, oflag; /* Command line flags. */
78     char *pname, *projidstr, *comment; /* projent fields. */
79     char *users, *groups, *attrs;
80     projid_t projid;
81
82     lst_t errlst;
83
84     /* Project file defaults to system project file "/etc/project" */
85     users = groups = comment = projidstr = "";
86
87     nflag = pflag = oflag = B_FALSE;
88     attrs = util_safe_zmalloc(1);
89     lst_create(&errlst);
90
91
92     (void) setlocale(LC_ALL, "");
93 #if !defined(TEXT_DOMAIN) /* Should be defined by cc -D */
94 #define TEXT_DOMAIN "SYS_TEST" /* Use this only if it wasn't */
95 #endif
96     (void) textdomain(TEXT_DOMAIN);
97
98     /* Parse the command line argument list */
99     while ((c = getopt(argc, argv, "hnfp:oc:U:G:K:")) != EOF)
100     switch (c) {
101         case 'h':
102             usage();
103             exit(0);
104             break;
105         case 'n':
106             nflag = B_TRUE;
107             break;
108         case 'f':
109             projfile = optarg;
110             break;
111         case 'p':
112             pflag = B_TRUE;
113             projidstr = optarg;
114             break;
115         case 'o':
116             oflag = B_TRUE;
117             break;
118         case 'c':
119             comment = optarg;
120             break;
121         case 'U':
122             users = optarg;
123             break;
124         case 'G':
125             groups = optarg;
126             break;
127         case 'K':
128             break;
129     }
130
131     if (errlst != NULL) {
132         if (lst_is_empty(errlst))
133             free(errlst);
134         else
135             util_print_errmsgs(errlst);
136     }
137
138     if (plist != NULL)
139         projent_free_lst(plist);
140
141     if (attrs != NULL)
142         free(attrs);
143
144     if (projfile != PROJF_PATH)
145         free(projfile);
146
147     if (users != NULL)
148         free(users);
149
150     if (groups != NULL)
151         free(groups);
152
153     if (comment != NULL)
154         free(comment);
155
156     if (projidstr != NULL)
157         free(projidstr);
158
159     if (nflag)
160         projent_set_nflag(ent);
161
162     if (pflag)
163         projent_set_projfile(ent, projfile);
164
165     if (oflag)
166         projent_set_oflag(ent);
167
168     if (users)
169         projent_set_users(ent, users);
170
171     if (groups)
172         projent_set_groups(ent, groups);
173
174     if (comment)
175         projent_set_comment(ent, comment);
176
177     if (projidstr)
178         projent_set_projidstr(ent, projidstr);
179
180     if (ent)
181         projent_free(ent);
182
183     if (errlst)
184         free(errlst);
185
186     if (plist)
187         free(plist);
188
189     if (attrs)
190         free(attrs);
191
192     if (projfile)
193         free(projfile);
194
195     if (users)
196         free(users);
197
198     if (groups)
199         free(groups);
200
201     if (comment)
202         free(comment);
203
204     if (projidstr)
205         free(projidstr);
206
207     if (ent)
208         projent_free(ent);
209
210     if (errlst)
211         free(errlst);
212
213     if (plist)
214         free(plist);
215
216     if (attrs)
217         free(attrs);
218
219     if (projfile)
220         free(projfile);
221
222     if (users)
223         free(users);
224
225     if (groups)
226         free(groups);
227
228     if (comment)
229         free(comment);
230
231     if (projidstr)
232         free(projidstr);
233
234     if (ent)
235         projent_free(ent);
236
237     if (errlst)
238         free(errlst);
239
240     if (plist)
241         free(plist);
242
243     if (attrs)
244         free(attrs);
245
246     if (projfile)
247         free(projfile);
248
249     if (users)
250         free(users);
251
252     if (groups)
253         free(groups);
254
255     if (comment)
256         free(comment);
257
258     if (projidstr)
259         free(projidstr);
260
261     if (ent)
262         projent_free(ent);
263
264     if (errlst)
265         free(errlst);
266
267     if (plist)
268         free(plist);
269
270     if (attrs)
271         free(attrs);
272
273     if (projfile)
274         free(projfile);
275
276     if (users)
277         free(users);
278
279     if (groups)
280         free(groups);
281
282     if (comment)
283         free(comment);
284
285     if (projidstr)
286         free(projidstr);
287
288     if (ent)
289         projent_free(ent);
290
291     if (errlst)
292         free(errlst);
293
294     if (plist)
295         free(plist);
296
297     if (attrs)
298         free(attrs);
299
300     if (projfile)
301         free(projfile);
302
303     if (users)
304         free(users);
305
306     if (groups)
307         free(groups);
308
309     if (comment)
310         free(comment);
311
312     if (projidstr)
313         free(projidstr);
314
315     if (ent)
316         projent_free(ent);
317
318     if (errlst)
319         free(errlst);
320
321     if (plist)
322         free(plist);
323
324     if (attrs)
325         free(attrs);
326
327     if (projfile)
328         free(projfile);
329
330     if (users)
331         free(users);
332
333     if (groups)
334         free(groups);
335
336     if (comment)
337         free(comment);
338
339     if (projidstr)
340         free(projidstr);
341
342     if (ent)
343         projent_free(ent);
344
345     if (errlst)
346         free(errlst);
347
348     if (plist)
349         free(plist);
350
351     if (attrs)
352         free(attrs);
353
354     if (projfile)
355         free(projfile);
356
357     if (users)
358         free(users);
359
360     if (groups)
361         free(groups);
362
363     if (comment)
364         free(comment);
365
366     if (projidstr)
367         free(projidstr);
368
369     if (ent)
370         projent_free(ent);
371
372     if (errlst)
373         free(errlst);
374
375     if (plist)
376         free(plist);
377
378     if (attrs)
379         free(attrs);
379
380     if (projfile)
381         free(projfile);
382
383     if (users)
384         free(users);
385
386     if (groups)
387         free(groups);
388
389     if (comment)
390         free(comment);
391
392     if (projidstr)
393         free(projidstr);
394
395     if (ent)
396         projent_free(ent);
397
398     if (errlst)
399         free(errlst);
399
400     if (plist)
401         free(plist);
402
403     if (attrs)
404         free(attrs);
405
406     if (projfile)
407         free(projfile);
408
409     if (users)
410         free(users);
411
412     if (groups)
413         free(groups);
414
415     if (comment)
416         free(comment);
417
418     if (projidstr)
419         free(projidstr);
420
421     if (ent)
422         projent_free(ent);
423
424     if (errlst)
425         free(errlst);
425
426     if (plist)
427         free(plist);
428
429     if (attrs)
430         free(attrs);
430
431     if (projfile)
431         free(projfile);
432
433     if (users)
434         free(users);
435
436     if (groups)
437         free(groups);
438
439     if (comment)
440         free(comment);
441
442     if (projidstr)
443         free(projidstr);
444
445     if (ent)
446         projent_free(ent);
447
448     if (errlst)
449         free(errlst);
449
450     if (plist)
451         free(plist);
452
453     if (attrs)
454         free(attrs);
454
455     if (projfile)
455         free(projfile);
456
457     if (users)
458         free(users);
459
460     if (groups)
461         free(groups);
462
463     if (comment)
464         free(comment);
465
466     if (projidstr)
467         free(projidstr);
468
469     if (ent)
470         projent_free(ent);
471
472     if (errlst)
473         free(errlst);
473
474     if (plist)
475         free(plist);
476
477     if (attrs)
478         free(attrs);
478
479     if (projfile)
480         free(projfile);
481
482     if (users)
483         free(users);
484
485     if (groups)
486         free(groups);
487
488     if (comment)
489         free(comment);
489
490     if (projidstr)
491         free(projidstr);
492
493     if (ent)
494         projent_free(ent);
495
496     if (errlst)
497         free(errlst);
497
498     if (plist)
499         free(plist);
499
499     if (attrs)
500         free(attrs);
500
501     if (projfile)
501         free(projfile);
502
503     if (users)
504         free(users);
505
506     if (groups)
507         free(groups);
508
509     if (comment)
510         free(comment);
510
511     if (projidstr)
512         free(projidstr);
513
514     if (ent)
515         projent_free(ent);
516
517     if (errlst)
518         free(errlst);
518
519     if (plist)
520         free(plist);
520
521     if (attrs)
522         free(attrs);
522
523     if (projfile)
523         free(projfile);
524
525     if (users)
526         free(users);
527
528     if (groups)
529         free(groups);
529
530     if (comment)
531         free(comment);
531
532     if (projidstr)
533         free(projidstr);
534
535     if (ent)
536         projent_free(ent);
537
538     if (errlst)
539         free(errlst);
539
540     if (plist)
541         free(plist);
541
542     if (attrs)
543         free(attrs);
543
544     if (projfile)
545         free(projfile);
545
546     if (users)
547         free(users);
548
549     if (groups)
550         free(groups);
550
551     if (comment)
552         free(comment);
552
553     if (projidstr)
554         free(projidstr);
555
556     if (ent)
557         projent_free(ent);
558
559     if (errlst)
559         free(errlst);
559
560     if (plist)
561         free(plist);
561
562     if (attrs)
563         free(attrs);
563
564     if (projfile)
565         free(projfile);
565
566     if (users)
567         free(users);
568
569     if (groups)
570         free(groups);
570
571     if (comment)
572         free(comment);
572
573     if (projidstr)
574         free(projidstr);
575
576     if (ent)
577         projent_free(ent);
578
579     if (errlst)
579         free(errlst);
579
580     if (plist)
581         free(plist);
581
582     if (attrs)
583         free(attrs);
583
584     if (projfile)
585         free(projfile);
585
586     if (users)
587         free(users);
588
589     if (groups)
590         free(groups);
590
591     if (comment)
592         free(comment);
592
593     if (projidstr)
594         free(projidstr);
595
596     if (ent)
597         projent_free(ent);
598
599     if (errlst)
599         free(errlst);
599
600     if (plist)
601         free(plist);
601
602     if (attrs)
603         free(attrs);
603
604     if (projfile)
605         free(projfile);
605
606     if (users)
607         free(users);
608
609     if (groups)
610         free(groups);
610
611     if (comment)
612         free(comment);
612
613     if (projidstr)
614         free(projidstr);
615
616     if (ent)
617         projent_free(ent);
618
619     if (errlst)
619         free(errlst);
619
620     if (plist)
621         free(plist);
621
622     if (attrs)
623         free(attrs);
623
624     if (projfile)
625         free(projfile);
625
626     if (users)
627         free(users);
628
629     if (groups)
630         free(groups);
630
631     if (comment)
632         free(comment);
632
633     if (projidstr)
634         free(projidstr);
635
636     if (ent)
637         projent_free(ent);
638
639     if (errlst)
639         free(errlst);
639
640     if (plist)
641         free(plist);
641
642     if (attrs)
643         free(attrs);
643
644     if (projfile)
645         free(projfile);
645
646     if (users)
647         free(users);
648
649     if (groups)
650         free(groups);
650
651     if (comment)
652         free(comment);
652
653     if (projidstr)
654         free(projidstr);
655
656     if (ent)
657         projent_free(ent);
658
659     if (errlst)
659         free(errlst);
659
660     if (plist)
661         free(plist);
661
662     if (attrs)
663         free(attrs);
663
664     if (projfile)
665         free(projfile);
665
666     if (users)
667         free(users);
668
669     if (groups)
670         free(groups);
670
671     if (comment)
672         free(comment);
672
673     if (projidstr)
674         free(projidstr);
675
676     if (ent)
677         projent_free(ent);
678
679     if (errlst)
679         free(errlst);
679
680     if (plist)
681         free(plist);
681
682     if (attrs)
683         free(attrs);
683
684     if (projfile)
685         free(projfile);
685
686     if (users)
687         free(users);
688
689     if (groups)
690         free(groups);
690
691     if (comment)
692         free(comment);
692
693     if (projidstr)
694         free(projidstr);
695
696     if (ent)
697         projent_free(ent);
698
699     if (errlst)
699         free(errlst);
699
700     if (plist)
701         free(plist);
701
702     if (attrs)
703         free(attrs);
703
704     if (projfile)
705         free(projfile);
705
706     if (users)
707         free(users);
708
709     if (groups)
710         free(groups);
710
711     if (comment)
712         free(comment);
712
713     if (projidstr)
714         free(projidstr);
715
716     if (ent)
717         projent_free(ent);
718
719     if (errlst)
719         free(errlst);
719
720     if (plist)
721         free(plist);
721
722     if (attrs)
723         free(attrs);
723
724     if (projfile)
725         free(projfile);
725
726     if (users)
727         free(users);
728
729     if (groups)
730         free(groups);
730
731     if (comment)
732         free(comment);
732
733     if (projidstr)
734         free(projidstr);
735
736     if (ent)
737         projent_free(ent);
738
739     if (errlst)
739         free(errlst);
739
740     if (plist)
741         free(plist);
741
742     if (attrs)
743         free(attrs);
743
744     if (projfile)
745         free(projfile);
745
746     if (users)
747         free(users);
748
749     if (groups)
750         free(groups);
750
751     if (comment)
752         free(comment);
752
753     if (projidstr)
754         free(projidstr);
755
756     if (ent)
757         projent_free(ent);
758
759     if (errlst)
759         free(errlst);
759
760     if (plist)
761         free(plist);
761
762     if (attrs)
763         free(attrs);
763
764     if (projfile)
765         free(projfile);
765
766     if (users)
767         free(users);
768
769     if (groups)
770         free(groups);
770
771     if (comment)
772         free(comment);
772
773     if (projidstr)
774         free(projidstr);
775
776     if (ent)
777         projent_free(ent);
778
779     if (errlst)
779         free(errlst);
779
780     if (plist)
781         free(plist);
781
782     if (attrs)
783         free(attrs);
783
784     if (projfile)
785         free(projfile);
785
786     if (users)
787         free(users);
788
789     if (groups)
790         free(groups);
790
791     if (comment)
792         free(comment);
792
793     if (projidstr)
794         free(projidstr);
795
796     if (ent)
797         projent_free(ent);
798
799     if (errlst)
799         free(errlst);
799
800     if (plist)
801         free(plist);
801
802     if (attrs)
803         free(attrs);
803
804     if (projfile)
805         free(projfile);
805
806     if (users)
807         free(users);
808
809     if (groups)
810         free(groups);
810
811     if (comment)
812         free(comment);
812
813     if (projidstr)
814         free(projidstr);
815
816     if (ent)
817         projent_free(ent);
818
819     if (errlst)
819         free(errlst);
819
820     if (plist)
821         free(plist);
821
822     if (attrs)
823         free(attrs);
823
824     if (projfile)
825         free(projfile);
825
826     if (users)
827         free(users);
828
829     if (groups)
830         free(groups);
830
831     if (comment)
832         free(comment);
832
833     if (projidstr)
834         free(projidstr);
835
836     if (ent)
837         projent_free(ent);
838
839     if (errlst)
839         free(errlst);
839
840     if (plist)
841         free(plist);
841
842     if (attrs)
843         free(attrs);
843
844     if (projfile)
845         free(projfile);
845
846     if (users)
847         free(users);
848
849     if (groups)
850         free(groups);
850
851     if (comment)
852         free(comment);
852
853     if (projidstr)
854         free(projidstr);
855
856     if (ent)
857         projent_free(ent);
858
859     if (errlst)
859         free(errlst);
859
860     if (plist)
861         free(plist);
861
862     if (attrs)
863         free(attrs);
863
864     if (projfile)
865         free(projfile);
865
866     if (users)
867         free(users);
868
869     if (groups)
870         free(groups);
870
871     if (comment)
872         free(comment);
872
873     if (projidstr)
874         free(projidstr);
875
876     if (ent)
877         projent_free(ent);
878
879     if (errlst)
879         free(errlst);
879
880     if (plist)
881         free(plist);
881
882     if (attrs)
883         free(attrs);
883
884     if (projfile)
885         free(projfile);
885
886     if (users)
887         free(users);
888
889     if (groups)
890         free(groups);
890
891     if (comment)
892         free(comment);
892
893     if (projidstr)
894         free(projidstr);
895
896     if (ent)
897         projent_free(ent);
898
899     if (errlst)
899         free(errlst);
899
900     if (plist)
901         free(plist);
901
902     if (attrs)
903         free(attrs);
903
904     if (projfile)
905         free(projfile);
905
906     if (users)
907         free(users);
908
909     if (groups)
910         free(groups);
910
911     if (comment)
912         free(comment);
912
913     if (projidstr)
914         free(projidstr);
915
916     if (ent)
917         projent_free(ent);
918
919     if (errlst)
919         free(errlst);
919
920     if (plist)
921         free(plist);
921
922     if (attrs)
923         free(attrs);
923
924     if (projfile)
925         free(projfile);
925
926     if (users)
927         free(users);
928
929     if (groups)
930         free(groups);
930
931     if (comment)
932         free(comment);
932
933     if (projidstr)
934         free(projidstr);
935
936     if (ent)
937         projent_free(ent);
938
939     if (errlst)
939         free(errlst);
939
940     if (plist)
941         free(plist);
941
942     if (attrs)
943         free(attrs);
943
944     if (projfile)
945         free(projfile);
945
946     if (users)
947         free(users);
948
949     if (groups)
950         free(groups);
950
951     if (comment)
952         free(comment);
952
953     if (projidstr)
954         free(projidstr);
955
956     if (ent)
957         projent_free(ent);
958
959     if (errlst)
959         free(errlst);
959
960     if (plist)
961         free(plist);
961
962     if (attrs)
963         free(attrs);
963
964     if (projfile)
965         free(projfile);
965
966     if (users)
967         free(users);
968
969     if (groups)
970         free(groups);
970
971     if (comment)
972         free(comment);
972
973     if (projidstr)
974         free(projidstr);
975
976     if (ent)
977         projent_free(ent);
978
979     if (errlst)
979         free(errlst);
979
980     if (plist)
981         free(plist);
981
982     if (attrs)
983         free(attrs);
983
984     if (projfile)
985         free(projfile);
985
986     if (users)
987         free(users);
988
989     if (groups)
990         free(groups);
990
991     if (comment)
992         free(comment);
992
993     if (projidstr)
994         free(projidstr);
995
996     if (ent)
997         projent_free(ent);
998
999     if (errlst)
999         free(errlst);
999
1000    if (plist)
1001        free(plist);
1001
1002    if (attrs)
1003        free(attrs);
1003
1004    if (projfile)
1005        free(projfile);
1005
1006    if (users)
1007        free(users);
1008
1009    if (groups)
1010        free(groups);
1010
1011    if (comment)
1012        free(comment);
1012
1013    if (projidstr)
1014        free(projidstr);
1015
1016    if (ent)
1017        projent_free(ent);
1018
1019    if (errlst)
1019        free(errlst);
1019
1020    if (plist)
1021        free(plist);
1021
1022    if (attrs)
1023        free(attrs);
1023
1024    if (projfile)
1025        free(projfile);
1025
1026    if (users)
1027        free(users);
1028
1029    if (groups)
1030        free(groups);
1030
1031    if (comment)
1032        free(comment);
1032
1033    if (projidstr)
1034        free(projidstr);
1035
1036    if (ent)
1037        projent_free(ent);
1038
1039    if (errlst)
1039        free(errlst);
1039
1040    if (plist)
1041        free(plist);
1041
1042    if (attrs)
1043        free(attrs);
1043
1044    if (projfile)
1045        free(projfile);
1045
1046    if (users)
1047        free(users);
1048
1049    if (groups)
1050        free(groups);
1050
1051    if (comment)
1052        free(comment);
1052
1053    if (projidstr)
1054        free(projidstr);
1055
1056    if (ent)
1057        projent_free(ent);
1058
1059    if (errlst)
1059        free(errlst);
1059
1060    if (plist)
1061        free(plist);
1061
1062    if (attrs)
1063        free(attrs);
1063
1064    if (projfile)
1065        free(projfile);
1065
1066    if (users)
1067        free(users);
1068
1069    if (groups)
1070        free(groups);
1070
1071    if (comment)
1072        free(comment);
1072
1073    if (projidstr)
1074        free(projidstr);
1075
1076    if (ent)
1077        projent_free(ent);
1078
1079    if (errlst)
1079        free(errlst);
1079
1080    if (plist)
1081        free(plist);
1081
1082    if (attrs)
1083        free(attrs);
1083
1084    if (projfile)
1085        free(projfile);
1085
1086    if (users)
1087        free(users);
1088
1089    if (groups)
1090        free(groups);
1090
1091    if (comment)
1092        free(comment);
1092
1093    if (projidstr)
1094        free(projidstr);
1095
1096    if (ent)
1097        projent_free(ent);
1098
1099    if (errlst)
1099        free(errlst);
1099
1100    if (plist)
1101        free(plist);
1101
1102    if (attrs)
1103        free(attrs);
1103
1104    if (projfile)
1105        free(projfile);
1105
1106    if (users)
1107        free(users);
1108
1109    if (groups)
1110        free(groups);
1110
1111    if (comment)
1112        free(comment);
1112
1113    if (projidstr)
1114        free(projidstr);
1115
1116    if (ent)
1117        projent_free(ent);
1118
1119    if (errlst)
1119        free(errlst);
1119
1120    if (plist)
1121        free(plist);
1121
1122    if (attrs)
1123        free(attrs);
1123
1124    if (projfile)
1125        free(projfile);
1125
1126    if (users)
1127        free(users);
1128
1129    if (groups)
1130        free(groups);
1130
1131    if (comment)
1132        free(comment);
1132
1133    if (projidstr)
1134        free(projidstr);
1135
1136    if (ent)
1137        projent_free(ent);
1138
1139    if (errlst)
1139        free(errlst);
1139
1140    if (plist)
1141        free(plist);
1141
1142    if (attrs)
1143        free(attrs);
1143
1144    if (projfile)
1145        free(projfile);
1145
1146    if (users)
1147        free(users);
1148
1149    if (groups)
1150        free(groups);
1150
1151    if (comment)
1152        free(comment);
1152
1153    if (projidstr)
1154        free(projidstr);
1155
1156    if (ent)
1157        projent_free(ent);
1158
1159    if (errlst)
1159        free(errlst);
1159
1160    if (plist)
1161        free(plist);
1161
1162    if (attrs)
1163        free(attrs);
1163
1164    if (projfile)
1165        free(projfile);
1165
1166    if (users)
1167        free(users);
1168
1169    if (groups)
1170        free(groups);
1170
1171    if (comment)
1172        free(comment);
1172
1173    if (projidstr)
1174        free(projidstr);
1175
1176    if (ent)
1177        projent_free(ent);
1178
1179    if (errlst)
1179        free(errlst);
1179
1180    if (plist)
1181        free(plist);
1181
1182    if (attrs)
1183        free(attrs);
1183
1184    if (projfile)
1185        free(projfile);
1185
1186    if (users)
1187        free(users);
1188
1189    if (groups)
1190        free(groups);
1190
1191    if (comment)
1192        free(comment);
1192
1193    if (projidstr)
1194        free(projidstr);
1195
1196    if (ent)
1197        projent_free(ent);
1198
1199    if (errlst)
1199        free(errlst);
1199
1200    if (plist)
1201        free(plist);
1201
1202    if (attrs)
1203        free(attrs);
1203
1204    if (projfile)
1205        free(projfile);
1205
1206    if (users)
1207        free(users);
1208
1209    if (groups)
1210        free(groups);
1210
1211    if (comment)
1212        free(comment);
1212
1213    if (projidstr)
1214        free(projidstr);
1215
1216    if (ent)
1217        projent_free(ent);
1218
1219    if (errlst)
1219        free(errlst);
1219
1220    if (plist)
1221        free(plist);
1221
1222    if (attrs)
1223        free(attrs);
1223
1224    if (projfile)
1225        free(projfile);
1225
1226    if (users)
1227        free(users);
1228
1229    if (groups)
1230        free(groups);
1230
1231    if (comment)
1232        free(comment);
1232
1233    if (projidstr)
1234        free(projidstr);
1235
1236    if (ent)
1237        projent_free(ent);
1238
1239    if (errlst)
1239        free(errlst);
1239
1240    if (plist)
1241        free(plist);
1241
1242    if (attrs)
1243        free(attrs);
1243
1244    if (projfile)
1245        free(projfile);
1245
1246    if (users)
1247        free(users);
1248
1249    if (groups)
1250        free(groups);
1250
1251    if (comment)
1252        free(comment);
1252
1253    if (projidstr)
1254        free(projidstr);
1255
1256    if (ent)
1257        projent_free(ent);
1258
1259    if (errlst)
1259        free(errlst);
1259
1260    if (plist)
1261        free(plist);
1261
1262    if (attrs)
1263        free(attrs);
1263
1264    if (projfile)
1265        free(projfile);
1265
1266    if (users)
1267        free(users
```

```

128             attrs = UTIL_STR_APPEND2(attrs, ";", optarg);
129             break;
130         default:
131             util_add_errmsg(&errlst, gettext(
132                     "Invalid option: -%c"), optopt);
133             break;
134     }

138     if (oflag && !pflag) {
139         util_add_errmsg(&errlst, gettext(
140                     "-o requires -p projid to be specified"));
141     }
143     if (optind != argc - 1) {
144         util_add_errmsg(&errlst, gettext("No project name specified"));
145     }
147     CHECK_ERRORS_FREE_PLST(&errlst, plist, attrs, 2);

149     if (!nflag)
150         flags |= F_PAR_VLD;
151     flags |= F_PAR_RES | F_PAR_DUP;

153     /* Parse the project file to get the list of the projects */
154     plist = projent_get_lst(projfile, flags, &errlst);
155     CHECK_ERRORS_FREE_PLST(&errlst, plist, attrs, 2);

158     /* Parse and validate new project id */
159     if (pflag && projent_parse_projid(projidstr, &projid, &errlst) == 0) {
160         if (!nflag) {
161             projent_validate_projid(projid, 0, &errlst);
162             if (!oflag) {
163                 projent_validate_unique_id(plist, projid,
164                     &errlst);
165             }
166         }
167     }
168     CHECK_ERRORS_FREE_PLST(&errlst, plist, attrs, 2);

170     /* Find the maxprojid */
171     for (e = 0; e < lst_size(plist); e++) {
172         ent = lst_at(plist, e);
173         maxpjid = (ent->projid > maxpjid) ? ent->projid : maxpjid;
174     }

177     if (!pflag && asprintf(&projidstr, "%ld", maxpjid + 1) == -1) {
178         util_add_errmsg(&errlst, gettext("Failed to allocate memory"));
179         CHECK_ERRORS_FREE_PLST(&errlst, plist, attrs, 2);
180     }

182     pname = argv[optind];
183     ent = projent_parse_components(pname, projidstr, comment, users,
184         groups, attrs, F_PAR_SPC | F_PAR_UNT, &errlst);
185     if (!pflag)
186         free(projidstr);

188     if (!nflag)
189         projent_validate_unique_name(plist, pname, &errlst);

191     CHECK_ERRORS_FREE_PLST(&errlst, plist, attrs, 2);

193     /* Sort attributes list */

```

```

194     projent_sort_attributes(ent->attrs);

197     /* Add the new project entry to the list */
198     lst_insert_tail(plst, ent);

200     /* Validate the projent before writing the list to the project file */
201     (void) projent_validate(ent, flags, &errlist);
202     CHECK_ERRORS_FREE_PLST(&errlst, plist, attrs, 2);

204     /* Write out the project file */
205     projent_put_lst(projfile, plist, &errlst);
206     CHECK_ERRORS_FREE_PLST(&errlst, plist, attrs, 2);

208 }
209 }
```

```
new/usr/src/cmd/projadd/projdel/Makefile
```

```
*****  
1722 Thu Dec 15 19:51:09 2016  
new/usr/src/cmd/projadd/projdel/Makefile  
Want projadd, projdel and projmod in C.  
*****
```

```
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 2009 Sun Microsystems, Inc. All rights reserved.  
23 # Use is subject to license terms.  
24 #  
  
26 PROG = projdel  
27 OBJS = projdel.o  
28 SRCS =$(OBJS:%.o=%.c) $(COMMON_SRCS)  
  
30 include $(SRC)/cmd/Makefile.cmd  
31 include $(SRC)/cmd/projadd/Makefile.projadd  
  
33 LDLIBS += -lpool  
34 CFLAGS += $(CCVERBOSE) -I${PROJADDCOMMONDIR}  
35 CERRWARN += -_gcc=-Wno-uninitialized  
36 CERRWARN += -_gcc=-Wno-switch  
37 CERRWARN += -_gcc=-Wno-parentheses  
  
39 CPPFLAGS_sparc += -I$(SRC)/uts/sfmmu  
40 CPPFLAGS_sparc += -I$(SRC)/uts/sun4u/sunfire  
41 CPPFLAGS += $(CPPFLAGS_${MACH})  
  
43 FILEMODE= 0555  
  
45 lint := LINTFLAGS = -muxs -I${PROJADDCOMMONDIR}  
  
47 .KEEP_STATE:  
  
49 all: $(PROG)  
  
51 install: all $(ROOTPROG)  
  
53 $(PROG): $(OBJS) $(COMMON_OBJS)  
54     $(LINK.c) -o $(PROG) $(OBJS) $(COMMON_OBJS) $(LDLIBS)  
55     $(POST_PROCESS)  
  
57 %.o : ${PROJADDCOMMONDIR}/%.c  
58     $(COMPILE.c) -o $@ $<  
59     $(POST_PROCESS_O)  
  
61 clean:
```

```
1
```

```
new/usr/src/cmd/projadd/projdel/Makefile
```

```
62      -$(RM) $(PROG) $(OBJS) $(COMMON_OBJS)  
64 lint: lint_SRCS  
66 include $(SRC)/cmd/Makefile.targ
```

```
2
```

```
new/usr/src/cmd/projadd/projdel/projdel.c
```

```
*****
```

```
3275 Thu Dec 15 19:51:10 2016
```

```
new/usr/src/cmd/projadd/projdel/projdel.c
```

```
Want projadd, projdel and projmod in C.
```

```
*****
```

```
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 #include <stdio.h>  
23 #include <stdlib.h>  
24 #include <libintl.h>  
25 #include <locale.h>  
26 #include <errno.h>  
27 #include <string.h>  
28 #include <stddef.h>  
29 #include <sys/types.h>  
  
31 #include "projent.h"  
32 #include "util.h"  
  
34 #define SEQU(str1, str2) (strcmp(str1, str2) == 0)  
  
36 /*  
37  * Print usage  
38  */  
39 static void  
40 usage(void)  
41 {  
42     (void) fprintf(stderr, gettext(  
43         "Usage:\n"  
44         "projdel [-f filename] project\n"));  
45 }  
  
47 /*  
48  * main()  
49  */  
50 int  
51 main(int argc, char **argv)  
52 {  
53     int e, c, ret = 0;  
54     int flags;  
55     extern char *optarg;  
56     extern int optind, optopt;  
57     lst_t *plst; /* Projects list */  
58     projent_t *ent, *delent;  
59     int del;  
60     char *pname; /* Project name */  
61     lst_t errlst; /* Errors list */
```

```
1
```

```
new/usr/src/cmd/projadd/projdel/projdel.c
```

```
62     char *projfile = PROJF_PATH; /* Project file "/etc/project" */  
64     lst_create(&errlst);  
  
67     (void) setlocale(LC_ALL, "");  
68 #if !defined(TEXT_DOMAIN) /* Should be defined by cc -D */  
69 #define TEXT_DOMAIN "SYS_TEST" /* Use this only if it wasn't */  
70 #endif  
71     (void) textdomain(TEXT_DOMAIN);  
  
73     /* Parse the command line argument list */  
74     while ((c = getopt(argc, argv, ":hf:")) != EOF)  
75         switch (c) {  
76             case 'h':  
77                 usage();  
78                 exit(0);  
79                 break;  
80             case 'f':  
81                 projfile = optarg;  
82                 break;  
83             default:  
84                 util_add_errmsg(&errlst, gettext(  
85                     "Invalid option: -%c"), optopt);  
86                 break;  
87         }  
88     if (optind != argc - 1) {  
89         (void) fprintf(stderr, gettext("No project name specified\n"));  
90         exit(2);  
91     }  
92     /* Name of the project to delete */  
93     pname = argv[optind];  
94     flags = F_PAR_VLD | F_PAR_RES | F_PAR_DUP;  
95     /* Parse the project file to get the list of the projects */  
96     plist = projent_get_lst(projfile, flags, &errlst);  
97     if (!lst_is_empty(&errlst)) {  
98         util_print_errmsgs(&errlst);  
99         usage();  
100        exit(2);  
101    }  
102    /* Find the project to be deleted */  
103    del = 0;  
104    for (e = 0; e < lst_size(plist); e++) {  
105        ent = lst_at(plist, e);  
106        if (SEQU(ent->projname, pname)) {  
107            del++;  
108            delent = ent;  
109        }  
110    }  
111    if (del == 0) {  
112        (void) fprintf(stderr, gettext(  
113            "Project \"%s\" does not exist\n"), pname);  
114        usage();  
115        ret = 2;  
116        goto out;  
117    } else if (del > 1) {  
118        (void) fprintf(stderr, gettext(  
119            "Duplicate project name \"%s\"\n"), pname);  
120        usage();  
121    }  
122    /* Delete the project */  
123    lst_delete(plist, delent);  
124    /* Free the project entry */  
125    free(delent);  
126    /* Free the project list */  
127    lst_free(plist);
```

```
2
```

```
128         ret = 2;
129         goto out;
130     }
132 
133     /* Remove the project entry from the list */
134     (void) lst_remove(plst, delent);
135 
136     /* Write out the project file */
137     projent_put_lst(projfile, plst, &errlst);
138 
139     if (!lst_is_empty(&errlst)) {
140         util_print_errmsgs(&errlst);
141         usage();
142         ret = 2;
143     }
144 
145     projent_free_lst(plst);
146     free(plst);
147     return (ret);
```

```
new/usr/src/cmd/projadd/projmod/Makefile
```

```
*****  
1732 Thu Dec 15 19:51:10 2016  
new/usr/src/cmd/projadd/projmod/Makefile  
Want projadd, projdel and projmod in C.  
*****
```

```
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 2009 Sun Microsystems, Inc. All rights reserved.  
23 # Use is subject to license terms.  
24 #  
  
26 PROG = projmod  
27 OBJS = projmod.o  
28 SRCS =$(OBJS:%.o=%.c) $(COMMON_SRCS)  
  
30 include $(SRC)/cmd/Makefile.cmd  
31 include $(SRC)/cmd/projadd/Makefile.projadd  
  
33 LDLIBS += -lpool -lproject  
34 CFLAGS += $(CCVERBOSE) -I${PROJADDCOMMONDIR}  
35 CERRWARN += -_gcc=-Wno-uninitialized  
36 CERRWARN += -_gcc=-Wno-switch  
37 CERRWARN += -_gcc=-Wno-parentheses  
  
39 CPPFLAGS_sparc += -I$(SRC)/uts/sfmmu  
40 CPPFLAGS_sparc += -I$(SRC)/uts/sun4u/sunfire  
41 CPPFLAGS += $(CPPFLAGS_${MACH})  
  
43 FILEMODE= 0555  
  
45 lint := LINTFLAGS = -muxs -I$(PROJADDCOMMONDIR)  
  
47 .KEEP_STATE:  
  
49 all: $(PROG)  
  
51 install: all $(ROOTPROG)  
  
53 $(PROG): $(OBJS) $(COMMON_OBJS)  
54     $(LINK.c) -o $(PROG) $(OBJS) $(COMMON_OBJS) $(LDLIBS)  
55     $(POST_PROCESS)  
  
57 %.o : $(PROJADDCOMMONDIR)/%.c  
58     $(COMPILE.c) -o $@ $<  
59     $(POST_PROCESS_O)  
  
61 clean:
```

```
1
```

```
new/usr/src/cmd/projadd/projmod/Makefile
```

```
62      -$(RM) $(PROG) $(OBJS) $(COMMON_OBJS)  
64 lint: lint_SRCS  
66 include $(SRC)/cmd/Makefile.targ
```

```
2
```

```
new/usr/src/cmd/projadd/projmod/projmod.c
```

```
*****
9347 Thu Dec 15 19:51:10 2016
new/usr/src/cmd/projadd/projmod/projmod.c
Want projadd, projdel and projmod in C.
*****
```

```
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 #include <stdio.h>
23 #include <stdlib.h>
24 #include <libintl.h>
25 #include <locale.h>
26 #include <errno.h>
27 #include <string.h>
28 #include <stddef.h>
29 #include <sys/types.h>
30 #include <sys/task.h>
31
32 #include <sys/debug.h>
33
34 #include "projent.h"
35 #include "util.h"
36
37 #define SEQU(str1, str2)          (strcmp(str1, str2) == 0)
38
39 #define CHECK_ERRORS_FREE_PLST(errlst, plist, attrs, ecode) { \
40     if (!lst_is_empty(errlst)) { \
41         util_print_errmsgs(errlst); \
42         if (plist != NULL) { \
43             projent_free_lst(plist); \
44             free(plist); \
45         } \
46         free(attrs); \
47         usage(); \
48         exit(ecode); \
49     } \
50 }
51
52
53 /*
54  * Print usage
55  */
56 */
57 static void
58 usage(void)
59 {
60     (void) fprintf(stderr, gettext(
61         "Usage:\n"
```

```
1
```

```
new/usr/src/cmd/projadd/projmod/projmod.c
```

```
62         "projmod [-n] [-A|-f filename] [-p projid [-o]] [-c comment]\n"
63         "           [-a|-s|-r] [-U user[,user...]] [-G group[,group...]]\n"
64         "           [-K name=value[,value...]] [-l new_projectname]\n"
65         "           project\n");
66 }
67
68 /*
69  * main()
70  */
71 int
72 main(int argc, char **argv)
73 {
74     int e, c, error;
75
76     extern char *optarg;
77     extern int optind, optopt;
78     lst_t *plist = NULL;
79     int flags = 0;
80     projent_t *ent, *modent;
81
82     /* Command line options */
83     boolean_t fflag, nflag, cflag, oflag, pflag, lflag,
84     boolean_t sflag, rflag, aflag;
85     boolean_t Uflag, Gflag, Kflag, Aflag;
86     boolean_t modify;
87
88     /* Project entry fields */
89     char *pname, *npname;
90     char *comment, *users, *groups, *attrs;
91     char *pusers, *pgroups;
92
93     lst_t *pattrbs;
94
95     lst_t errlst;
96
97     /* Project file defaults to system project file "/etc/project" */
98     char *projfile = PROJF_PATH;
99     struct project proj, *projp;
100    char buf[PROJECT_BUFSZ];
101    char *str;
102
103    comment = users = groups = "";
104    pname = npname = NULL;
105
106    fflag = nflag = cflag = oflag = pflag = lflag = B_FALSE;
107    sflag = rflag = aflag = B_FALSE;
108    Uflag = Gflag = Kflag = Aflag = B_FALSE;
109
110    modify = B_FALSE;
111
112
113    attrs = util_safe_zmalloc(1);
114    lst_create(&errlst);
115
116
117    (void) setlocale(LC_ALL, "");
118    #if !defined(TEXT_DOMAIN) /* Should be defined by cc -D */
119    #define TEXT_DOMAIN "SYS_TEST" /* Use this only if it wasn't */
120    #endif
121    (void) textdomain(TEXT_DOMAIN);
122
123    /* Parse the command line argument list */
124    while ((c = getopt(argc, argv, "hf:nc:op:l:sraU:G:K:A")) != EOF)
125        switch (c) {
```

```
2
```

```

128         case 'h':
129             usage();
130             exit(0);
131             break;
132         case 'f':
133             fflag = B_TRUE;
134             projfile = optarg;
135             break;
136         case 'n':
137             nflag = B_TRUE;
138             break;
139         case 'c':
140             cflag = B_TRUE;
141             comment = optarg;
142             break;
143         case 'o':
144             oflag = B_TRUE;
145             break;
146         case 'p':
147             pflag = B_TRUE;
148             break;
149         case 'l':
150             lflag = B_TRUE;
151             npname = optarg;
152             break;
153         case 's':
154             sflag = B_TRUE;
155             break;
156         case 'r':
157             rflag = B_TRUE;
158             break;
159         case 'a':
160             aflag = B_TRUE;
161             break;
162         case 'U':
163             Uflag = B_TRUE;
164             users = optarg;
165             break;
166         case 'G':
167             Gflag = B_TRUE;
168             groups = optarg;
169             break;
170         case 'K':
171             Kflag = B_TRUE;
172             attrs = UTIL_STR_APPEND2(attrs, ";", optarg);
173             break;
174         case 'A':
175             Aflag = B_TRUE;
176             break;
177         default:
178             util_add_errmsg(&errlst, gettext(
179                 "Invalid option: -%c"), optopt);
180             break;
181     }
183     CHECK_ERRORS_FREE_PLST(&errlst, plst, attrs, 2);
185     if (optind == argc - 1) {
186         pname = argv[optind];
187     }
189     if (cflag || Gflag || lflag || pflag || Uflag || Kflag || Aflag) {
190         modify = B_TRUE;
191         if (pname == NULL) {
192             util_add_errmsg(&errlst, gettext(
193                 "No project name specified"));

```

```

194         }
195     } else if (pname != NULL) {
196         util_add_errmsg(&errlst, gettext(
197             "missing -c, -G, -l, -p, -U, or -K"));
198     }
199
200     if (Aflag && fflag) {
201         util_add_errmsg(&errlst, gettext(
202             "-A and -f are mutually exclusive"));
203     }
204
205     if (oflag && !pflag) {
206         util_add_errmsg(&errlst, gettext(
207             "-o requires -p projid to be specified"));
208     }
209
210     if ((aflag && (rflag || sflag)) || (rflag && (aflag || sflag)) ||
211         (sflag && (aflag || rflag))) {
212         util_add_errmsg(&errlst, gettext(
213             "-a, -r, and -s are mutually exclusive"));
214     }
215
216     if ((aflag || rflag || sflag) && !(Uflag || Gflag || Kflag)) {
217         util_add_errmsg(&errlst, gettext(
218             "-a, -r, and -s require -U users, -G groups "
219             "or -K attributes to be specified"));
220     }
221
222     CHECK_ERRORS_FREE_PLST(&errlst, plst, attrs, 2);
223
224     if (aflag) {
225         flags |= F_MOD_ADD;
226     } else if (rflag) {
227         flags |= F_MOD_DEL;
228     } else if (sflag) {
229         flags |= F_MOD_SUB;
230     } else {
231         flags |= F_MOD_REP;
232     }
233     if (!nflag) {
234         flags |= F_PAR_VLD;
235     }
236     flags |= F_PAR_RES | F_PAR_DUP;
237
238     plst = projent_get_lst(projfile, flags, &errlst);
239     CHECK_ERRORS_FREE_PLST(&errlst, plst, attrs, 2);
240
241     modent = NULL;
242     if (pname != NULL) {
243         for (e = 0; e < lst_size(plst); e++) {
244             ent = lst_at(plst, e);
245             if (SEQU(ent->projname, pname)) {
246                 modent = ent;
247             }
248         }
249         if (modent == NULL) {
250             util_add_errmsg(&errlst, gettext(
251                 "Project \"%s\" does not exist"), pname);
252         }
253     }
254
255     CHECK_ERRORS_FREE_PLST(&errlst, plst, attrs, 2);
256
257     /*
258      * If there is no modification options, simply reading the file, which
259      * includes parsing and verifying, is sufficient.

```

```

        util_add_errmsg(&errlst, gettext(
            "user \'%s\' is not a member "
            "of project \'%s\'", "root", pname));
    } else {
        util_add_errmsg(&errlst, gettext(
            "could not join project \'%s\'", pname));
    }
} else if (error == SETPROJ_ERR_POOL) {
if (errno == EACCES) {
    util_add_errmsg(&errlst, gettext(
        "no resource pool accepting default "
        "bindings exists for project \'%s\'",
        pname));
} else if (errno == ESRCRH) {
    util_add_errmsg(&errlst, gettext(
        "specified resource pool does not exist "
        "for project \'%s\'", pname));
} else {
    util_add_errmsg(&errlst, gettext(
        "could not bind to default resource pool "
        "for project \'%s\'", pname));
}
} else {
/*
 * error represents the position - within the
 * semi-colon delimited attribute - that generated
 * the error.
 */
if (error <= 0) {
    util_add_errmsg(&errlst, gettext(
        "setproject failed for project \'%s\'",
        pname));
} else {
    /* To be completed */
    projp = getprojbyname(pname, &proj, buf,
        sizeof (buf));
    pattribs = (projp != NULL) ?
        projent_parse_attributes(projp->pj_attr,
        0, &errlst) : NULL;
    if (projp != NULL && pattribs != NULL &&
        (str = projent_attrib_tostring(
        lst_at(pattribs, error - 1))) != NULL) {
        util_add_errmsg(&errlst, gettext(
            "warning, \"%s\" resource control "
            "assignment failed for project "
            "\"%s\"", str, pname));
        free(str);
    } else {
        util_add_errmsg(&errlst, gettext(
            "warning, resource control "
            "assignment failed for project "
            "\"%s\" attribute %d", pname,
            error));
    }
    if (pattribs != NULL) {
        projent_free_attributes(pattribs);
        UTIL_FREE_SNNULL(pattribs);
    }
}
}
ERRORS_FREE_PLST(&errlst, plst, attrs, 2);
()
;
```

392 }

```

new/usr/src/cmd/projadd/projtest/Makefile
*****
1724 Thu Dec 15 19:51:10 2016
new/usr/src/cmd/projadd/projtest/Makefile
Want projadd, projdel and projmod in C.
*****
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 2009 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24 #

26 PROG = projtest
27 OBJS = projtest.o
28 SRCS =$(OBJS:%.o=%.c) $(COMMON_SRCS)

30 include $(SRC)/cmd/Makefile.cmd
31 include $(SRC)/cmd/projadd/Makefile.projadd

33 LDLIBS += -lpool
34 CFLAGS += $(CCVERBOSE) -I${PROJADDCOMMONDIR}
35 CERRWARN += -_gcc=-Wno-uninitialized
36 CERRWARN += -_gcc=-Wno-switch
37 CERRWARN += -_gcc=-Wno-parentheses

39 CPPFLAGS_sparc += -I$(SRC)/uts/sfmmu
40 CPPFLAGS_sparc += -I$(SRC)/uts/sun4u/sunfire
41 CPPFLAGS += $(CPPFLAGS_${MACH})

43 FILEMODE= 0555

45 lint := LINTFLAGS = -muxs -I${PROJADDCOMMONDIR}

47 .KEEP_STATE:

49 all: $(PROG)

51 install: all $(ROOTPROG)

53 $(PROG): $(OBJS) $(COMMON_OBJS)
54     $(LINK.c) -o $(PROG) $(OBJS) $(COMMON_OBJS) $(LDLIBS)
55     $(POST_PROCESS)

57 %.o : ${PROJADDCOMMONDIR}/%.c
58     $(COMPILE.c) -o $@ $<
59     $(POST_PROCESS_O)

61 clean:

```

```

1
new/usr/src/cmd/projadd/projtest/Makefile
*****
62      -$(RM) $(PROG) $(OBJS) $(COMMON_OBJS)
64 lint: lint_SRCS
66 include $(SRC)/cmd/Makefile.targ
2

```