

```

*****
57841 Fri Jan 12 10:52:06 2018
new/usr/src/lib/fm/topo/libtopo/common/topo_xml.c
8954 libtopo cannot handle any array type other than string_array.
Reviewed by: Andy Stormont astormont@racktopsystems.com
Reviewed by: David H^ppner 0xfee@gmail.com
Reviewed by: Rob Johnston rob.johnston@joyent.com
*****
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 /*
23  * Copyright (c) 2006, 2010, Oracle and/or its affiliates. All rights reserved.
24  * Copyright (c) 2013, Joyent, Inc. All rights reserved.
25  * Copyright (c) 2018, Western Digital Technologies, Inc. All rights reserved.
26 */
27
28 #include <libxml/parser.h>
29 #include <libxml/xinclud.h>
30 #include <sys/fm/protocol.h>
31 #include <assert.h>
32 #include <strings.h>
33 #include <string.h>
34 #include <ctype.h>
35 #include <errno.h>
36 #include <limits.h>
37 #include <fm/libtopo.h>
38 #include <unistd.h>
39 #include <sys/stat.h>
40 #include <fcntl.h>
41 #include <topo_file.h>
42 #include <topo_mod.h>
43 #include <topo_subr.h>
44 #include <topo_alloc.h>
45 #include <topo_parse.h>
46 #include <topo_error.h>
47
48 static tf_rdata_t *topo_xml_walk(topo_mod_t *, tf_info_t *, xmlNodePtr,
49     tnode_t *);
50 static tf_edata_t *enum_attributes_process(topo_mod_t *, xmlNodePtr);
51 static int enum_run(topo_mod_t *, tf_rdata_t *);
52 static int fac_enum_run(topo_mod_t *, tnode_t *, const char *);
53 static int fac_process(topo_mod_t *, xmlNodePtr, tf_rdata_t *, tnode_t *);
54 static int fac_enum_process(topo_mod_t *, xmlNodePtr, tnode_t *);
55 static int decorate_nodes(topo_mod_t *, tf_rdata_t *, xmlNodePtr, tnode_t *,
56     tf_pad_t **);

```

```

59 static void
60 strarr_free(topo_mod_t *mod, char **arr, uint_t nelems)
61 {
62     int i;
63
64     for (i = 0; i < nelems; i++)
65         topo_mod_strfree(mod, arr[i]);
66     topo_mod_free(mod, arr, (nelems * sizeof (char *)));
67 }
68
69 unchanged portion omitted
70
71 196 static int
72 197 xlate_common(topo_mod_t *mp, xmlNodePtr xn, topo_type_t ptype, nvlist_t *nvl,
73 198     const char *name)
74 199 const char *name)
75 199 {
76 200     int rv;
77 201     uint64_t ui;
78 202     uint_t i = 0, nelems = 0;
79 203     nvlist_t *fmri;
80 204     xmlChar *str;
81 205     char **strarrbuf;
82 206     void *arrbuf;
83 207     nvlist_t **nvlarrbuf;
84 208     xmlNodePtr cn;
85
86 210     topo_dprintf(mp->tm_hdl, TOPO_DBG_XML, "xlate_common(name=%s)\n", name);
87 211     switch (ptype) {
88 212     case TOPO_TYPE_INT32:
89 213         if (xmlattr_to_int(mp, xn, Value, &ui) < 0)
90 214             return (-1);
91 215         rv = nvlist_add_int32(nvl, name, (int32_t)ui);
92 216         break;
93 217     case TOPO_TYPE_UINT32:
94 218         if (xmlattr_to_int(mp, xn, Value, &ui) < 0)
95 219             return (-1);
96 220         rv = nvlist_add_uint32(nvl, name, (uint32_t)ui);
97 221         break;
98 222     case TOPO_TYPE_INT64:
99 223         if (xmlattr_to_int(mp, xn, Value, &ui) < 0)
100 224             return (-1);
101 225         rv = nvlist_add_int64(nvl, name, (int64_t)ui);
102 226         break;
103 227     case TOPO_TYPE_UINT64:
104 228         if (xmlattr_to_int(mp, xn, Value, &ui) < 0)
105 229             return (-1);
106 230         rv = nvlist_add_uint64(nvl, name, ui);
107 231         break;
108 232     case TOPO_TYPE_FMRI:
109 233         if (xmlattr_to_fmri(mp, xn, Value, &fmri) < 0)
110 234             return (-1);
111 235         rv = nvlist_add_nvlist(nvl, name, fmri);
112 236         nvlist_free(fmri);
113 237         break;
114 238     case TOPO_TYPE_STRING:
115 239         if ((str = xmlGetProp(xn, (xmlChar *)Value)) == NULL)
116 240             return (-1);
117 241         rv = nvlist_add_string(nvl, name, (char *)str);
118 242         xmlFree(str);
119 243         break;
120 244     case TOPO_TYPE_INT32_ARRAY:
121 245     case TOPO_TYPE_UINT32_ARRAY:
122 246     case TOPO_TYPE_INT64_ARRAY:
123 247     case TOPO_TYPE_UINT64_ARRAY:
124 248         for (cn = xn->xmlChildrenNode; cn != NULL; cn = cn->next)
125             if ((xmlStrcmp(cn->name, (xmlChar *)Propitem) == 0) ||

```

```

249         (xmlStrcmp(cn->name, (xmlChar *)Argitem) == 0))
250         nelems++;
251
252     if (nelems < 1) {
253         topo_dprintf(mp->tm_hdl, TOPO_DBG_ERR, "No <propitem> "
254             "or <argitem> elements found for array val");
255         return (-1);
256     }
257     if ((arrbuf = topo_mod_alloc(mp, (nelems * sizeof (uint64_t))))
258         == NULL)
259         return (topo_mod_seterrno(mp, ETOPO_NOMEM));
260     break;
261 case TOPO_TYPE_STRING_ARRAY:
262 case TOPO_TYPE_FMRI_ARRAY:
263     for (cn = xn->xmlChildrenNode; cn != NULL; cn = cn->next)
264         if ((xmlStrcmp(cn->name, (xmlChar *)Propitem) == 0) ||
265             (xmlStrcmp(cn->name, (xmlChar *)Argitem) == 0))
266             nelems++;
267
268     if (nelems < 1) {
269         topo_dprintf(mp->tm_hdl, TOPO_DBG_ERR, "No <propitem> "
270             "or <argitem> elements found for array val");
271         return (-1);
272     }
273     if ((strarrbuf = topo_mod_alloc(mp, (nelems * sizeof (char *))))
274         == NULL)
275         return (topo_mod_seterrno(mp, ETOPO_NOMEM));
276     break;
277 case TOPO_TYPE_FMRI_ARRAY:
278     for (cn = xn->xmlChildrenNode; cn != NULL; cn = cn->next)
279         if ((xmlStrcmp(cn->name, (xmlChar *)Propitem) == 0) ||
280             (xmlStrcmp(cn->name, (xmlChar *)Argitem) == 0))
281             nelems++;
282
283     if (nelems < 1) {
284         topo_dprintf(mp->tm_hdl, TOPO_DBG_ERR, "No <propitem> "
285             "elements found for array prop");
286         return (-1);
287     }
288     if ((nvlarrbuf = topo_mod_alloc(mp, (nelems *
289         sizeof (nvlist_t *))) == NULL)
290         return (topo_mod_seterrno(mp, ETOPO_NOMEM));
291     break;
292 default:
293     topo_dprintf(mp->tm_hdl, TOPO_DBG_ERR,
294         "Unrecognized type attribute (ptype = %d)\n", ptype);
295     return (topo_mod_seterrno(mp, ETOPO_PRSR_BADTYPE));
296 }
297
298 switch (ptype) {
299 case TOPO_TYPE_INT32_ARRAY:
300     if ((arrbuf = topo_mod_alloc(mp, (nelems * sizeof (int32_t))))
301         == NULL)
302         return (topo_mod_seterrno(mp, ETOPO_NOMEM));
303     for (cn = xn->xmlChildrenNode; cn != NULL; cn = cn->next) {
304         if ((xmlStrcmp(cn->name, (xmlChar *)Propitem) == 0) ||
305             (xmlStrcmp(cn->name, (xmlChar *)Argitem) == 0)) {
306             if ((str = xmlGetProp(cn, (xmlChar *)Value))
307                 if ((str = xmlGetProp(xn, (xmlChar *)Value))
308                     == NULL)
309                 return (-1);
310
311             ((int32_t *)arrbuf)[i++]
312                 = atoi((const char *)str);
313             xmlFree(str);
314         }
315     }
316     rv = nvlist_add_int32_array(nvl, name, (int32_t *)arrbuf,
317         nelems);
318     topo_mod_free(mp, arrbuf, (nelems * sizeof (int32_t)));
319     free(arrbuf);
320     break;
321 case TOPO_TYPE_UINT32_ARRAY:
322     if ((arrbuf = topo_mod_alloc(mp, (nelems * sizeof (uint32_t))))
323         == NULL)
324         return (topo_mod_seterrno(mp, ETOPO_NOMEM));
325     for (cn = xn->xmlChildrenNode; cn != NULL; cn = cn->next) {
326         if ((xmlStrcmp(cn->name, (xmlChar *)Propitem) == 0) ||
327             (xmlStrcmp(cn->name, (xmlChar *)Argitem) == 0)) {
328             if ((str = xmlGetProp(cn, (xmlChar *)Value))
329                 if ((str = xmlGetProp(xn, (xmlChar *)Value))
330                     == NULL)
331                 return (-1);
332
333             ((uint32_t *)arrbuf)[i++]
334                 = atoi((const char *)str);
335             xmlFree(str);
336         }
337     }
338     rv = nvlist_add_uint32_array(nvl, name, (uint32_t *)arrbuf,
339         nelems);
340     topo_mod_free(mp, arrbuf, (nelems * sizeof (uint32_t)));
341     free(arrbuf);
342     break;
343 case TOPO_TYPE_INT64_ARRAY:
344     if ((arrbuf = topo_mod_alloc(mp, (nelems * sizeof (int64_t))))
345         == NULL)
346         return (topo_mod_seterrno(mp, ETOPO_NOMEM));
347     for (cn = xn->xmlChildrenNode; cn != NULL; cn = cn->next) {
348         if ((xmlStrcmp(cn->name, (xmlChar *)Propitem) == 0) ||
349             (xmlStrcmp(cn->name, (xmlChar *)Argitem) == 0)) {
350             if ((str = xmlGetProp(cn, (xmlChar *)Value))
351                 if ((str = xmlGetProp(xn, (xmlChar *)Value))
352                     == NULL)
353                 return (-1);
354
355             ((int64_t *)arrbuf)[i++]
356                 = atol((const char *)str);
357             xmlFree(str);
358         }
359     }
360     rv = nvlist_add_int64_array(nvl, name, (int64_t *)arrbuf,
361         nelems);
362     topo_mod_free(mp, arrbuf, (nelems * sizeof (int64_t)));
363     free(arrbuf);
364     break;
365 case TOPO_TYPE_UINT64_ARRAY:
366     if ((arrbuf = topo_mod_alloc(mp, (nelems * sizeof (uint64_t))))
367         == NULL)
368         return (topo_mod_seterrno(mp, ETOPO_NOMEM));
369     for (cn = xn->xmlChildrenNode; cn != NULL; cn = cn->next) {
370         if ((xmlStrcmp(cn->name, (xmlChar *)Propitem) == 0) ||
371             (xmlStrcmp(cn->name, (xmlChar *)Argitem) == 0)) {
372             if ((str = xmlGetProp(cn, (xmlChar *)Value))
373                 if ((str = xmlGetProp(xn, (xmlChar *)Value))
374                     == NULL)
375                 return (-1);
376
377             ((uint64_t *)arrbuf)[i++]
378                 = atol((const char *)str);
379             xmlFree(str);
380         }
381     }
382     rv = nvlist_add_uint64_array(nvl, name, (uint64_t *)arrbuf,
383         nelems);
384     topo_mod_free(mp, arrbuf, (nelems * sizeof (uint64_t)));
385     free(arrbuf);
386     break;
387 }

```

```

283     }
284 }
285
286 rv = nvlist_add_int32_array(nvl, name, (int32_t *)arrbuf,
287     nelems);
288 topo_mod_free(mp, arrbuf, (nelems * sizeof (int32_t)));
289 free(arrbuf);
290 break;
291 case TOPO_TYPE_UINT32_ARRAY:
292     if ((arrbuf = topo_mod_alloc(mp, (nelems * sizeof (uint32_t))))
293         == NULL)
294         return (topo_mod_seterrno(mp, ETOPO_NOMEM));
295     for (cn = xn->xmlChildrenNode; cn != NULL; cn = cn->next) {
296         if ((xmlStrcmp(cn->name, (xmlChar *)Propitem) == 0) ||
297             (xmlStrcmp(cn->name, (xmlChar *)Argitem) == 0)) {
298             if ((str = xmlGetProp(cn, (xmlChar *)Value))
299                 if ((str = xmlGetProp(xn, (xmlChar *)Value))
300                     == NULL)
301                 return (-1);
302
303             ((uint32_t *)arrbuf)[i++]
304                 = atoi((const char *)str);
305             xmlFree(str);
306         }
307     }
308     rv = nvlist_add_uint32_array(nvl, name, (uint32_t *)arrbuf,
309         nelems);
310     topo_mod_free(mp, arrbuf, (nelems * sizeof (uint32_t)));
311     free(arrbuf);
312     break;
313 case TOPO_TYPE_INT64_ARRAY:
314     if ((arrbuf = topo_mod_alloc(mp, (nelems * sizeof (int64_t))))
315         == NULL)
316         return (topo_mod_seterrno(mp, ETOPO_NOMEM));
317     for (cn = xn->xmlChildrenNode; cn != NULL; cn = cn->next) {
318         if ((xmlStrcmp(cn->name, (xmlChar *)Propitem) == 0) ||
319             (xmlStrcmp(cn->name, (xmlChar *)Argitem) == 0)) {
320             if ((str = xmlGetProp(cn, (xmlChar *)Value))
321                 if ((str = xmlGetProp(xn, (xmlChar *)Value))
322                     == NULL)
323                 return (-1);
324
325             ((int64_t *)arrbuf)[i++]
326                 = atol((const char *)str);
327             xmlFree(str);
328         }
329     }
330     rv = nvlist_add_int64_array(nvl, name, (int64_t *)arrbuf,
331         nelems);
332     topo_mod_free(mp, arrbuf, (nelems * sizeof (int64_t)));
333     free(arrbuf);
334     break;
335 case TOPO_TYPE_UINT64_ARRAY:
336     if ((arrbuf = topo_mod_alloc(mp, (nelems * sizeof (uint64_t))))
337         == NULL)
338         return (topo_mod_seterrno(mp, ETOPO_NOMEM));
339     for (cn = xn->xmlChildrenNode; cn != NULL; cn = cn->next) {
340         if ((xmlStrcmp(cn->name, (xmlChar *)Propitem) == 0) ||
341             (xmlStrcmp(cn->name, (xmlChar *)Argitem) == 0)) {
342             if ((str = xmlGetProp(cn, (xmlChar *)Value))
343                 if ((str = xmlGetProp(xn, (xmlChar *)Value))
344                     == NULL)
345                 return (-1);
346
347             ((uint64_t *)arrbuf)[i++]
348                 = atol((const char *)str);
349             xmlFree(str);
350         }
351     }
352     rv = nvlist_add_uint64_array(nvl, name, (uint64_t *)arrbuf,
353         nelems);
354     topo_mod_free(mp, arrbuf, (nelems * sizeof (uint64_t)));
355     free(arrbuf);
356     break;
357 }

```

```

343         == NULL)
344         return (-1);
345
346         ((uint64_t *)arrbuf)[i++]
347         = atol((const char *)str);
348         xmlFree(str);
349     }
350 }
351
352 rv = nvlist_add_uint64_array(nvl, name, arrbuf,
353     nelems);
354 topo_mod_free(mp, arrbuf, (nelems * sizeof (uint64_t)));
355 free(arrbuf);
356 break;
357 case TOPO_TYPE_STRING_ARRAY:
358     if ((strarrbuf = topo_mod_alloc(mp, (nelems * sizeof (char *))))
359         == NULL)
360         return (topo_mod_seterrno(mp, ETOPO_NOMEM));
361     for (cn = xn->xmlChildrenNode; cn != NULL; cn = cn->next) {
362         if ((xmlStrcmp(cn->name, (xmlChar *)Propitem) == 0) ||
363             (xmlStrcmp(cn->name, (xmlChar *)Argitem) == 0)) {
364             if ((str = xmlGetProp(cn, (xmlChar *)Value))
365                 == NULL)
366                 return (-1);
367
368             strarrbuf[i++] =
369                 topo_mod_strdup(mp, (const char *)str);
370             xmlFree(str);
371         }
372     }
373
374     rv = nvlist_add_string_array(nvl, name, strarrbuf, nelems);
375     strarr_free(mp, strarrbuf, nelems);
376     break;
377 case TOPO_TYPE_FMRI_ARRAY:
378     if ((nvlarrbuf = topo_mod_alloc(mp, (nelems *
379         sizeof (nvlist_t *))) == NULL)
380         return (topo_mod_seterrno(mp, ETOPO_NOMEM));
381     for (cn = xn->xmlChildrenNode; cn != NULL; cn = cn->next) {
382         if ((xmlStrcmp(cn->name, (xmlChar *)Propitem) == 0) ||
383             (xmlStrcmp(cn->name, (xmlChar *)Argitem) == 0)) {
384             if ((str = xmlGetProp(cn, (xmlChar *)Value))
385                 if ((str = xmlGetProp(xn, (xmlChar *)Value))
386                     == NULL)
387                 return (-1);
388
389             if (topo_mod_str2nvl(mp, (const char *)str,
390                 &(nvlarrbuf[i++])) < 0) {
391                 xmlFree(str);
392                 return (-1);
393             }
394             xmlFree(str);
395         }
396     }
397
398     rv = nvlist_add_nvlist_array(nvl, name, nvlarrbuf,
399     nelems);
400     topo_mod_free(mp, nvlarrbuf, (nelems * sizeof (nvlist_t *)));
401     free(nvlarrbuf);
402     break;
403 }
404
405 if (rv != 0) {
406     topo_dprintf(mp->tm_hdl, TOPO_DBG_ERR,

```

```

406         "Nvlist construction failed.\n");
407         return (topo_mod_seterrno(mp, ETOPO_NOMEM));
408     } else
409         return (0);
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```