```
**********************************************************
   43621 Tue Apr 23 06:04:04 2019
new/usr/src/lib/libctf/common/ctf_merge.c
10827 some symbols have the wrong CTF type
Reviewed by: Robert Mustacchi <rm@joyent.com>
**********************************************************
```

```
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */

/*
 * Copyright 2019, Joyent, Inc.
 * Copyright (c) 2019 Joyent, Inc.
 */

/*
 * To perform a merge of two CTF containers, we first diff the two containers
 * types. For every type that's in the src container, but not in the dst
 * container, we note it and add it to dst container. If there are any objects
 * or functions associated with src, we go through and update the types that
 * they refer to such that they all refer to types in the dst container.
 *
 * The bulk of the logic for the merge, after we've run the diff, occurs in
 * ctf_merge_common().
 *
 * In terms of exported APIs, we don't really export a simple merge two
 * containers, as the general way this is used, in something like ctfmerge(1),
 * is to add all the containers and then let us figure out the best way to merge
 * it.
 */

#include <libctf_impl.h>
#include <sys/debug.h>
#include <sys/list.h>
#include <stddef.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <mergeq.h>
#include <errno.h>

typedef struct ctf_merge_tinfo {
        uint16_t cmt_map;       /* Map to the type in out */
        boolean_t cmt_fixup;
        boolean_t cmt_forward;
        boolean_t cmt_missing;
} ctf_merge_tinfo_t;
```

*_____unchanged_portion_omitted_*

```
/*
 * Now that we've successfully merged everything, we're going to remap the type
 * table.
 *
 * Remember we have two containers: ->cm_src is what we're working from, and
 * ->cm_out is where we are building the de-duplicated CTF.
 *
 * The index of this table is always the type IDs in ->cm_src.
 *
 * When we built this table originally in ctf_diff_self(), if we found a novel
```



```
 * type, we marked it as .cmt_missing to indicate it needs adding to ->cm_out.
 * Otherwise, .cmt_map indicated the ->cm_src type ID that this type duplicates.
 *
 * Then, in ctf_merge_common(), we walked through and added all "cmt_missing"
 * types to ->cm_out with ctf_merge_add_type(). These routines update cmt_map
 * to be the *new* type ID in ->cm_out.  In this function, you can read
 * "cmt_missing" as meaning "added to ->cm_out, and cmt_map updated".
 *
 * So at this point, we need to mop up all types where .cmt_missing == B_FALSE,
 * making sure *their* .cmt_map values also point to the ->cm_out container.
 * Now that we've successfully merged everything, we're going to clean
 * up the merge type table. Traditionally if we had just two different
 * files that we were working between, the types would be fully
 * resolved. However, because we were comparing with ourself every step
 * of the way and not our reduced self, we need to go through and update
 * every mapped entry to what it now points to in the deduped file.
 */
static void
ctf_merge_dedup_remap(ctf_merge_types_t *cmp)
ctf_merge_fixup_dedup_map(ctf_merge_types_t *cmp)
{
        int i;

        for (i = 1; i < cmp->cm_src->ctf_typemax + 1; i++) {
                ctf_id_t tid;

                /*
                 * Missing types always have their id updated to exactly what it
                 * should be.
                 */
                if (cmp->cm_tmap[i].cmt_missing == B_TRUE) {
                        VERIFY(cmp->cm_tmap[i].cmt_map != 0);
                        continue;
                }

                tid = i;
                while (cmp->cm_tmap[tid].cmt_missing == B_FALSE) {
                        VERIFY(cmp->cm_tmap[tid].cmt_map != 0);
                        tid = cmp->cm_tmap[tid].cmt_map;
                }
                VERIFY(cmp->cm_tmap[tid].cmt_map != 0);
                cmp->cm_tmap[i].cmt_map = cmp->cm_tmap[tid].cmt_map;
        }
}


/*
 * We're going to do three passes over the containers.
 *
 * Pass 1 checks for forward references in the output container that we know
 * exist in the source container.
 *
 * Pass 2 adds all the missing types from the source container. As part of this
 * we may be adding a type as a forward reference that doesn't exist yet.
 * Any types that we encounter in this form, we need to add to a third pass.
 *
 * Pass 3 is the fixup pass. Here we go through and find all the types that were
 * missing in the first.
 *
 * Importantly, we *must* call ctf_update between the second and third pass,
 * otherwise several of the libctf functions will not properly find the data in
 * the container. If we're doing a dedup we also fix up the type mapping.
 */
static int
ctf_merge_common(ctf_merge_types_t *cmp)
{
```

```
682         int ret, i;

684         ctf_phase_dump(cmp->cm_src, "merge-common-src", NULL);
685         ctf_phase_dump(cmp->cm_out, "merge-common-dest", NULL);

687         /* Pass 1 */
688         for (i = 1; i <= cmp->cm_src->ctf_typemax; i++) {
689                 if (cmp->cm_tmap[i].cmt_forward == B_TRUE) {
690                         ret = ctf_merge_add_sou(cmp, i, B_TRUE);
691                         if (ret != 0) {
692                                 return (ret);
693                         }
694                 }
695         }

697         /* Pass 2 */
698         for (i = 1; i <= cmp->cm_src->ctf_typemax; i++) {
699                 if (cmp->cm_tmap[i].cmt_missing == B_TRUE) {
700                         ret = ctf_merge_add_type(cmp, i);
701                         if (ret != 0) {
702                                 ctf_dprintf("Failed to merge type %d\n", i);
703                                 return (ret);
704                         }
705                 }
706         }

708         ret = ctf_update(cmp->cm_out);
709         if (ret != 0)
710                 return (ret);

712         if (cmp->cm_dedup == B_TRUE) {
713                 ctf_merge_dedup_remap(cmp);
704                 ctf_merge_fixup_dedup_map(cmp);
714         }

716         ctf_dprintf("Beginning merge pass 3\n");
717         /* Pass 3 */
718         for (i = 1; i <= cmp->cm_src->ctf_typemax; i++) {
719                 if (cmp->cm_tmap[i].cmt_fixup == B_TRUE) {
720                         ret = ctf_merge_fixup_type(cmp, i);
721                         if (ret != 0)
722                                 return (ret);
723                 }
724         }

717         if (cmp->cm_dedup == B_TRUE) {
718                 ctf_merge_fixup_dedup_map(cmp);
719         }

726         return (0);
727 }
```

*unchanged_portion_omitted_*