```
**********************************************************
    9364 Thu Jan 17 14:29:24 2019
new/usr/src/lib/libfru/libnvfru/nvfru.c
10102 libnvfru needs smatch fixes
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright (c) 2014 Gary Mills
 *
 * Copyright (c) 2009, 2010, Oracle and/or its affiliates. All rights reserved.
 *
 * Copyright (c) 2018, Joyent, Inc.
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <strings.h>
#include <assert.h>
#include <pthread.h>
#include <sys/byteorder.h>
#include <sys/types.h>
#include <sys/nvpair.h>

#include "libfru.h"
#include "libfrup.h"
#include "fru_tag.h"
#include "libfrureg.h"
#include "nvfru.h"

#define NUM_ITER_BYTES  4
#define HEAD_ITER       0
#define TAIL_ITER       1
#define NUM_ITER        2
#define MAX_ITER        3
#define TIMESTRINGLEN   128

#define PARSE_TIME      1

static pthread_mutex_t gLock = PTHREAD_MUTEX_INITIALIZER;



static void
convert_field(const uint8_t *field, const fru_regdef_t *def, const char *path,
    nvlist_t *nv)
```

```
{
        char timestring[TIMESTRINGLEN];
        int i;
        uint64_t value;
        time_t timefield;

        switch (def->dataType) {
        case FDTYPE_Binary:
                assert(def->payloadLen <= sizeof (value));
                switch (def->dispType) {
#if PARSE_TIME == 1
                case FDISP_Time:
                        if (def->payloadLen > sizeof (timefield)) {
                                /* too big for formatting */
                                return;
                        }
                        (void) memcpy(&timefield, field, sizeof (timefield));
                        timefield = BE_32(timefield);
                        if (strftime(timestring, sizeof (timestring), "%c",
                            localtime(&timefield)) == 0) {
                                /* buffer too small */
                                return;
                        }
                        (void) nvlist_add_string(nv, path, timestring);
                        return;
#endif

                case FDISP_Binary:
                case FDISP_Octal:
                case FDISP_Decimal:
                case FDISP_Hex:
                default:
                        value = 0;
                        (void) memcpy((((uint8_t *)&value) +
                            sizeof (value) - def->payloadLen),
                            field, def->payloadLen);
                        value = BE_64(value);
                        switch (def->payloadLen) {
                        case 1:
                                (void) nvlist_add_uint8(nv, path,
                                    (uint8_t)value);
                                break;
                        case 2:
                                (void) nvlist_add_uint16(nv, path,
                                    (uint16_t)value);
                                break;
                        case 4:
                                (void) nvlist_add_uint32(nv, path,
                                    (uint32_t)value);
                                break;
                        default:
                                (void) nvlist_add_uint64(nv, path, value);
                        }
                        return;
                }

        case FDTYPE_ASCII:
                (void) nvlist_add_string(nv, path, (char *)field);
                return;

        case FDTYPE_Enumeration:
                value = 0;
                (void) memcpy((((uint8_t *)&value) + sizeof (value) -
                    def->payloadLen), field, def->payloadLen);
                value = BE_64(value);
                for (i = 0; i < def->enumCount; i++) {
                        if (def->enumTable[i].value == value) {
                                (void) nvlist_add_string(nv, path,
                                    def->enumTable[i].text);
                                return;
                        }
                }
        }

        /* nothing matched above, use byte array */
        (void) nvlist_add_byte_array(nv, path, (uchar_t *)field,
            def->payloadLen);
}
```
______***unchanged_portion_omitted_***

```
int
rawfru_to_nvlist(uint8_t *buffer, size_t bufsize, char *cont_type,
    nvlist_t **nvlist)
{
        fru_errno_t fru_err;
        fru_nodehdl_t hdl;
        int err;

        (void) pthread_mutex_lock(&gLock);
        fru_err = fru_open_data_source("raw", buffer, bufsize, cont_type,
            NULL);
        if (fru_err != FRU_SUCCESS) {
                (void) pthread_mutex_unlock(&gLock);
                return (-1);
        }
        fru_err = fru_get_root(&hdl);
        if (fru_err != FRU_SUCCESS) {
                (void) pthread_mutex_unlock(&gLock);
                return (-1);
        }

        err = convert_fru(hdl, nvlist);

        (void) fru_close_data_source();
        fru_close_data_source();

        (void) pthread_mutex_unlock(&gLock);

        return (err);
}
```
______***unchanged_portion_omitted_***