```
**********************************************************
    7252 Thu Feb 20 13:54:46 2014
new/usr/src/lib/libdhcpagent/common/dhcp_stable.c
4586 dhcpv6 client id malformed
Reviewed by: Sebastien Roy <sebastien.roy@delphix.com>
Reviewed by: Marcel Telka <marcel@telka.sk>
**********************************************************
```

```
/*
 * 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 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * This module reads and writes the stable identifier values, DUID and IAID.
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <limits.h>
#include <fcntl.h>
#include <errno.h>
#include <libdlpi.h>
#include <uuid/uuid.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <net/if.h>
#include <netinet/dhcp6.h>
#include <dhcp_inittab.h>

#define DUID_FILE       "/etc/dhcp/duid"
#define IAID_FILE       "/etc/dhcp/iaid"

struct iaid_ent {
        uint32_t        ie_iaid;
        char            ie_name[LIFNAMSIZ];
};
_____unchanged_portion_omitted_

/*
 * make_stable_duid(): create a new DUID
 *
```



```
 *   input: const char *: name of physical interface for reference
 *          size_t *: pointer to a size_t to return the DUID length
 *  output: uchar_t *: the DUID buffer, or NULL on error (and errno is set)
 *    note: memory returned is from malloc; caller must free.
 */

uchar_t *
make_stable_duid(const char *physintf, size_t *duidlen)
{
        int len;
        dlpi_info_t dlinfo;
        dlpi_handle_t dh = NULL;
        uint_t arptype;
        duid_en_t *den;

        /*
         * Try to read the MAC layer address for the physical interface
         * provided as a hint.  If that works, we can use a DUID-LLT.
         */

        if (dlpi_open(physintf, &dh, 0) == DLPI_SUCCESS &&
            dlpi_bind(dh, DLPI_ANY_SAP, NULL) == DLPI_SUCCESS &&
#endif /* ! codereview */
            dlpi_info(dh, &dlinfo, 0) == DLPI_SUCCESS &&
            (len = dlinfo.di_physaddrlen) > 0 &&
            (arptype = dlpi_arptype(dlinfo.di_mactype) != 0)) {
                duid_llt_t *dllt;
                time_t now;

                if ((dllt = malloc(sizeof (*dllt) + len)) == NULL) {
                        dlpi_close(dh);
                        return (NULL);
                }

                (void) memcpy((dllt + 1), dlinfo.di_physaddr, len);
                dllt->dllt_dutype = htons(DHCPV6_DUID_LLT);
                dllt->dllt_hwtype = htons(arptype);
                now = time(NULL) - DUID_TIME_BASE;
                dllt->dllt_time = htonl(now);
                *duidlen = sizeof (*dllt) + len;
                dlpi_close(dh);
                return ((uchar_t *)dllt);
        }
        if (dh != NULL)
                dlpi_close(dh);

        /*
         * If we weren't able to create a DUID based on the network interface
         * in use, then generate one based on a UUID.
         */
        den = malloc(sizeof (*den) + UUID_LEN);
        if (den != NULL) {
                uuid_t uuid;

                den->den_dutype = htons(DHCPV6_DUID_EN);
                DHCPV6_SET_ENTNUM(den, DHCPV6_SUN_ENT);
                uuid_generate(uuid);
                (void) memcpy(den + 1, uuid, UUID_LEN);
                *duidlen = sizeof (*den) + UUID_LEN;
        }
        return ((uchar_t *)den);
}

/*
 * read_stable_iaid(): read a link's stable IAID, if any
 *
```

```
 *   input: const char *: interface name
 *  output: uint32_t: the IAID, or 0 if none
 */

uint32_t
read_stable_iaid(const char *intf)
{
        int fd;
        struct iaid_ent ie;

        if ((fd = open(IAID_FILE, O_RDONLY)) == -1)
                return (0);
        while (read(fd, &ie, sizeof (ie)) == sizeof (ie)) {
                if (strcmp(intf, ie.ie_name) == 0) {
                        (void) close(fd);
                        return (ie.ie_iaid);
                }
        }
        (void) close(fd);
        return (0);
}

/*
 * write_stable_iaid(): write out a link's stable IAID
 *
 *   input: const char *: interface name
 *  output: uint32_t: the IAID, or 0 if none
 */

int
write_stable_iaid(const char *intf, uint32_t iaid)
{
        int fd;
        struct iaid_ent ie;
        ssize_t retv;

        if ((fd = open(IAID_FILE, O_RDWR | O_CREAT, 0644)) == -1)
                return (0);
        while (read(fd, &ie, sizeof (ie)) == sizeof (ie)) {
                if (strcmp(intf, ie.ie_name) == 0) {
                        (void) close(fd);
                        if (iaid == ie.ie_iaid) {
                                return (0);
                        } else {
                                errno = EINVAL;
                                return (-1);
                        }
                }
        }
        (void) memset(&ie, 0, sizeof (ie));
        ie.ie_iaid = iaid;
        (void) strlcpy(ie.ie_name, intf, sizeof (ie.ie_name));
        retv = write(fd, &ie, sizeof (ie));
        (void) close(fd);
        if (retv == sizeof (ie)) {
                return (0);
        } else {
                if (retv >= 0)
                        errno = ENOSPC;
                return (-1);
        }
}

/*
 * make_stable_iaid(): create a stable IAID for a link
 *
```



```
 *   input: const char *: interface name
 *          uint32_t: the ifIndex for this link (as a "hint")
 *  output: uint32_t: the new IAID, never zero
 */

/* ARGSUSED */
uint32_t
make_stable_iaid(const char *intf, uint32_t hint)
{
        int fd;
        struct iaid_ent ie;
        uint32_t maxid, minunused;
        boolean_t recheck;

        if ((fd = open(IAID_FILE, O_RDONLY)) == -1)
                return (hint);
        maxid = 0;
        minunused = 1;
        /*
         * This logic is deliberately unoptimized.  The reason is that it runs
         * essentially just once per interface for the life of the system.
         * Once the IAID is established, there's no reason to generate it
         * again, and all we care about here is correctness.  Also, IAIDs tend
         * to get added in a logical sequence order, so the outer loop should
         * not normally run more than twice.
         */
        do {
                recheck = B_FALSE;
                while (read(fd, &ie, sizeof (ie)) == sizeof (ie)) {
                        if (ie.ie_iaid > maxid)
                                maxid = ie.ie_iaid;
                        if (ie.ie_iaid == minunused) {
                                recheck = B_TRUE;
                                minunused++;
                        }
                        if (ie.ie_iaid == hint)
                                hint = 0;
                }
                if (recheck)
                        (void) lseek(fd, 0, SEEK_SET);
        } while (recheck);
        (void) close(fd);
        if (hint != 0)
                return (hint);
        else if (maxid != UINT32_MAX)
                return (maxid + 1);
        else
                return (minunused);
}
```

```
**********************************************************
   47791 Thu Feb 20 13:54:47 2014
new/usr/src/lib/libdhcputil/common/dhcp_inittab.c
4586 dhcpv6 client id malformed
Reviewed by: Sebastien Roy <sebastien.roy@delphix.com>
Reviewed by: Marcel Telka <marcel@telka.sk>
**********************************************************
______unchanged_portion_omitted_

 455 /*
 456  * get_mac_addr(): interpret ",macaddr" in the input string, as part of a DUID.
 457  *                 The 'macaddr' may be a hex string (in any standard format),
 458  *                 or the name of a physical interface.  If an interface name
 459  *                 is given, then the interface type is extracted as well.
 460  *
 461  *   input: const char *: input string
 462  *          int *: error return value
 463  *          uint16_t *: hardware type output (network byte order)
 464  *          int: hardware type input; -1 for empty
 465  *          uchar_t *: output buffer for MAC address
 466  *  output: int: length of MAC address, or -1 for error
 467  */

 469 static int
 470 get_mac_addr(const char *str, int *ierrnop, uint16_t *hwret, int hwtype,
 471     uchar_t *outbuf)
 472 {
 473         int maclen;
 474         int dig, val;
 475         dlpi_handle_t dh;
 476         dlpi_info_t dlinfo;
 477         char chr;

 479         if (*str != '\0') {
 480                 if (*str++ != ',')
 481                         goto failed;
 482                 if (dlpi_open(str, &dh, 0) != DLPI_SUCCESS) {
 483                         maclen = 0;
 484                         dig = val = 0;
 485                         /*
 486                          * Allow MAC addresses with separators matching regexp
 487                          * (:|-| *).
 488                          */
 489                         while ((chr = *str++) != '\0') {
 490                                 if (isdigit(chr)) {
 491                                         val = (val << 4) + chr - '0';
 492                                 } else if (isxdigit(chr)) {
 493                                         val = (val << 4) + chr -
 494                                             (isupper(chr) ? 'A' : 'a') + 10;
 495                                 } else if (isspace(chr) && dig == 0) {
 496                                         continue;
 497                                 } else if (chr == ':' || chr == '-' ||
 498                                     isspace(chr)) {
 499                                         dig = 1;
 500                                 } else {
 501                                         goto failed;
 502                                 }
 503                                 if (++dig == 2) {
 504                                         *outbuf++ = val;
 505                                         maclen++;
 506                                         dig = val = 0;
 507                                 }
 508                         }
 509                 } else {
 510                         if (dlpi_bind(dh, DLPI_ANY_SAP, NULL) !=
 511                             DLPI_SUCCESS || dlpi_info(dh, &dlinfo, 0) !=
```



```
 512                             DLPI_SUCCESS) {
 510                         if (dlpi_info(dh, &dlinfo, 0) != DLPI_SUCCESS) {
 513                                 dlpi_close(dh);
 514                                 goto failed;
 515                         }
 516                         maclen = dlinfo.di_physaddrlen;
 517                         (void) memcpy(outbuf, dlinfo.di_physaddr, maclen);
 518                         dlpi_close(dh);
 519                         if (hwtype == -1)
 520                                 hwtype = dlpi_arptype(dlinfo.di_mactype);
 521                 }
 522         }
 523         if (hwtype == -1)
 524                 goto failed;
 525         *hwret = htons(hwtype);
 526         return (maclen);

 528 failed:
 529         *ierrnop = ITAB_BAD_NUMBER;
 530         return (-1);
 531 }
______unchanged_portion_omitted_
```