```
**********************************************************
   25655 Mon Jan 14 13:17:17 2019
new/usr/src/uts/common/io/ixgbe/ixgbe_gld.c
10086 ixgbe_m_propinfo() is missing an 'else'
**********************************************************
```

```
/*
 * 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) 2007-2010 Intel Corporation. All rights reserved.
 */

/*
 * Copyright (c) 2008, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
 * Copyright 2016 OmniTI Computer Consulting, Inc. All rights reserved.
 * Copyright (c) 2018, Joyent, Inc.
 * Copyright (c) 2017, Joyent, Inc.
 */

#include "ixgbe_sw.h"

/*
 * Bring the device out of the reset/quiesced state that it
 * was in when the interface was registered.
 */
int
ixgbe_m_start(void *arg)
{
        ixgbe_t *ixgbe = (ixgbe_t *)arg;

        mutex_enter(&ixgbe->gen_lock);

        if (ixgbe->ixgbe_state & IXGBE_SUSPENDED) {
                mutex_exit(&ixgbe->gen_lock);
                return (ECANCELED);
        }

        if (ixgbe_start(ixgbe, B_TRUE) != IXGBE_SUCCESS) {
                mutex_exit(&ixgbe->gen_lock);
                return (EIO);
        }

        atomic_or_32(&ixgbe->ixgbe_state, IXGBE_STARTED);

        mutex_exit(&ixgbe->gen_lock);

        /*
```



```
         * Enable and start the watchdog timer
         */
        ixgbe_enable_watchdog_timer(ixgbe);

        return (0);
}
```

unchanged_portion_omitted_

```
void
ixgbe_m_propinfo(void *arg, const char *pr_name, mac_prop_id_t pr_num,
    mac_prop_info_handle_t prh)
{
        ixgbe_t *ixgbe = (ixgbe_t *)arg;
        struct ixgbe_hw *hw = &ixgbe->hw;
        uint_t perm;
        uint8_t value;
        ixgbe_link_speed speeds = 0;

        /*
         * We cannot always rely on the common code maintaining
         * hw->phy.speeds_supported, therefore we fall back to use the
         * recorded supported speeds which were obtained during instance init in
         * ixgbe_init_params().
         */
        speeds = hw->phy.speeds_supported;
        if (speeds == 0)
                speeds = ixgbe->speeds_supported;

        switch (pr_num) {
        case MAC_PROP_DUPLEX:
        case MAC_PROP_SPEED:
                mac_prop_info_set_perm(prh, MAC_PROP_PERM_READ);
                break;

        case MAC_PROP_ADV_100FDX_CAP:
                mac_prop_info_set_perm(prh, MAC_PROP_PERM_READ);
                value = (speeds & IXGBE_LINK_SPEED_100_FULL) ? 1 : 0;
                mac_prop_info_set_default_uint8(prh, value);
                break;

        case MAC_PROP_ADV_1000FDX_CAP:
                mac_prop_info_set_perm(prh, MAC_PROP_PERM_READ);
                value = (speeds & IXGBE_LINK_SPEED_1GB_FULL) ? 1 : 0;
                mac_prop_info_set_default_uint8(prh, value);
                break;

        case MAC_PROP_ADV_2500FDX_CAP:
                mac_prop_info_set_perm(prh, MAC_PROP_PERM_READ);
                value = (speeds & IXGBE_LINK_SPEED_2_5GB_FULL) ? 1 : 0;
                mac_prop_info_set_default_uint8(prh, value);
                break;

        case MAC_PROP_ADV_5000FDX_CAP:
                mac_prop_info_set_perm(prh, MAC_PROP_PERM_READ);
                value = (speeds & IXGBE_LINK_SPEED_5GB_FULL) ? 1 : 0;
                mac_prop_info_set_default_uint8(prh, value);
                break;

        case MAC_PROP_ADV_10GFDX_CAP:
                mac_prop_info_set_perm(prh, MAC_PROP_PERM_READ);
                value = (speeds & IXGBE_LINK_SPEED_10GB_FULL) ? 1 : 0;
                mac_prop_info_set_default_uint8(prh, value);
                break;

        /*
         * We allow speed changes only on baseT PHYs. MAC_PROP_EN_* are marked
```

```
736          * read-only on non-baseT (SFP) PHYs.
737          */
738         case MAC_PROP_AUTONEG:
739                 perm = (hw->phy.media_type == ixgbe_media_type_copper) ?
740                     MAC_PROP_PERM_RW : MAC_PROP_PERM_READ;
741                 mac_prop_info_set_perm(prh, perm);
742                 mac_prop_info_set_default_uint8(prh, 1);
743                 break;
744
745         case MAC_PROP_EN_10GFDX_CAP:
746                 if (speeds & IXGBE_LINK_SPEED_10GB_FULL) {
747                         perm = (hw->phy.media_type == ixgbe_media_type_copper) ?
748                             MAC_PROP_PERM_RW : MAC_PROP_PERM_READ;
749                         mac_prop_info_set_perm(prh, perm);
750                         mac_prop_info_set_default_uint8(prh, 1);
751                 }
752                 break;
753
754         case MAC_PROP_EN_5000FDX_CAP:
755                 if (speeds & IXGBE_LINK_SPEED_5GB_FULL) {
756                         perm = (hw->phy.media_type == ixgbe_media_type_copper) ?
757                             MAC_PROP_PERM_RW : MAC_PROP_PERM_READ;
758                         mac_prop_info_set_perm(prh, perm);
759                         mac_prop_info_set_default_uint8(prh, 1);
760                 }
761                 break;
762
763         case MAC_PROP_EN_2500FDX_CAP:
764                 if (speeds & IXGBE_LINK_SPEED_2_5GB_FULL) {
765                         perm = (hw->phy.media_type == ixgbe_media_type_copper) ?
766                             MAC_PROP_PERM_RW : MAC_PROP_PERM_READ;
767                         mac_prop_info_set_perm(prh, perm);
768                         mac_prop_info_set_default_uint8(prh, 1);
769                 }
770                 break;
771
772         case MAC_PROP_EN_1000FDX_CAP:
773                 if (speeds & IXGBE_LINK_SPEED_1GB_FULL) {
774                         perm = (hw->phy.media_type == ixgbe_media_type_copper) ?
775                             MAC_PROP_PERM_RW : MAC_PROP_PERM_READ;
776                         mac_prop_info_set_perm(prh, perm);
777                         mac_prop_info_set_default_uint8(prh, 1);
778                 }
779                 break;
780
781         case MAC_PROP_EN_100FDX_CAP:
782                 if (speeds & IXGBE_LINK_SPEED_100_FULL) {
783                         perm = (hw->phy.media_type == ixgbe_media_type_copper) ?
784                             MAC_PROP_PERM_RW : MAC_PROP_PERM_READ;
785                         mac_prop_info_set_perm(prh, perm);
786                         mac_prop_info_set_default_uint8(prh, 1);
787                 }
788                 break;
789
790         case MAC_PROP_FLOWCTRL:
791                 mac_prop_info_set_default_link_flowctrl(prh,
792                     LINK_FLOWCTRL_NONE);
793                 break;
794
795         case MAC_PROP_MTU:
796                 mac_prop_info_set_range_uint32(prh,
797                     DEFAULT_MTU, ixgbe->capab->max_mtu);
798                 break;
799
800         case MAC_PROP_PRIVATE: {
801                 char valstr[64];
```



```
802                 int value;
803
804                 bzero(valstr, sizeof (valstr));
805
806                 if (strcmp(pr_name, "_adv_pause_cap") == 0 ||
807                     strcmp(pr_name, "_adv_asym_pause_cap") == 0) {
808                         mac_prop_info_set_perm(prh, MAC_PROP_PERM_READ);
809                         return;
810                 }
811
812                 if (strcmp(pr_name, "_tx_copy_thresh") == 0) {
813                         value = DEFAULT_TX_COPY_THRESHOLD;
814                 } else if (strcmp(pr_name, "_tx_recycle_thresh") == 0) {
815                         value = DEFAULT_TX_RECYCLE_THRESHOLD;
816                 } else if (strcmp(pr_name, "_tx_overload_thresh") == 0) {
817                         value = DEFAULT_TX_OVERLOAD_THRESHOLD;
818                 } else if (strcmp(pr_name, "_tx_resched_thresh") == 0) {
819                         value = DEFAULT_TX_RESCHED_THRESHOLD;
820                 } else  if (strcmp(pr_name, "_rx_copy_thresh") == 0) {
821                         value = DEFAULT_RX_COPY_THRESHOLD;
822                 } else  if (strcmp(pr_name, "_rx_limit_per_intr") == 0) {
823                         value = DEFAULT_RX_LIMIT_PER_INTR;
824                 } else if (strcmp(pr_name, "_intr_throttling") == 0) {
824                 }       if (strcmp(pr_name, "_intr_throttling") == 0) {
825                         value = ixgbe->capab->def_intr_throttle;
826                 } else {
827                         return;
828                 }
829
830                 (void) snprintf(valstr, sizeof (valstr), "%x", value);
831         }
832         }
833 }
```

*unchanged_portion_omitted_*