```
**********************************************************
    9871 Mon Aug 17 09:15:39 2015
new/usr/src/uts/common/sys/scsi/conf/device.h
6131 struct scsi_device uses a 1-bit signed bitfield
**********************************************************
/*
 * 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 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright 2014 Garrett D'Amore <garrett@damore.org>
 */

/*
 * SCSI device structure.
 *
 * All SCSI target drivers will have one of these per target/lun/sfunc.
 * It is allocated and initialized by the framework SCSA HBA nexus code
 * for each SCSI target dev_info_t node during HBA nexus DDI_CTLOPS_INITCHILD
 * processing of a child device node just prior to tran_tgt_init(9E).  A
 * pointer the the scsi_device(9S) structure is stored in the
 * driver-private data field of the target device's dev_info_t node (in
 * 'devi_driver_data') and can be retrieved by ddi_get_driver_private(9F).
 */
#ifndef _SYS_SCSI_CONF_DEVICE_H
#define _SYS_SCSI_CONF_DEVICE_H

#include <sys/scsi/scsi_types.h>

#ifdef  __cplusplus
extern "C" {
#endif

struct scsi_device {
        /*
         * Routing information for a SCSI device (target/lun/sfunc).
         *
         * The scsi_address(9S) structure contains a pointer to the
         * scsi_hba_tran(9S) of the transport.
         *
         * For devices below an HBA that uses SCSI_HBA_ADDR_SPI
         * unit-addressing, the scsi_address(9S) information contains
         * decoded target/lun addressing information.
         *
         * For devices below an HBA that uses SCSI_HBA_ADDR_COMPLEX
         * unit-addressing, the scsi_address(9S) information contains a
```

```
         * pointer to the scsi_device(9S) structure and the HBA can maintain
         * its private per-unit-address/per-scsi_device information using
         * scsi_address_device(9F) and scsi_device_hba_private_[gs]et(9F).
         *
         * NOTE: The scsi_address(9S) structure gets structure-copied into
         * the scsi_pkt(9S) 'pkt_address' field. Having a pointer to the
         * scsi_device(9S) structure within the scsi_address(9S) allows
         * the SCSA framework to reflect generic changes in device state
         * at scsi_pkt_comp(9F) time (given just a scsi_pkt(9S) pointer).
         *
         * NOTE: The older SCSI_HBA_TRAN_CLONE method of supporting
         * SCSI-3 devices is still supported, but use is discouraged.
         */
        struct scsi_address     sd_address;

        /* Cross-reference to target device's dev_info_t. */
        dev_info_t              *sd_dev;

        /*
         * Target driver mutex for this device. Initialized by SCSA HBA
         * framework code prior to probe(9E) or attach(9E) of scsi_device.
         */
        kmutex_t                sd_mutex;

        /*
         * SCSA private: use is associated with implementation of
         * SCSI_HBA_ADDR_COMPLEX scsi_device_hba_private_[gs]et(9F).
         * The HBA driver can store a pointer to per-scsi_device(9S)
         * HBA private data during its tran_tgt_init(9E) implementation
         * by calling scsi_device_hba_private_set(9F), and free that
         * pointer during tran_tgt_fini(9E). At tran_send(9E) time, the
         * HBA driver can use scsi_address_device(9F) to obtain a pointer
         * to the scsi_device(9S) structure, and then gain access to
         * its per-scsi_device(9S) hba private data by calling
         * scsi_device_hba_private_get(9F).
         */
        void                    *sd_hba_private;

        /*
         * If scsi_slave is used to probe out this device, a scsi_inquiry data
         * structure will be allocated and an INQUIRY command will be run to
         * fill it in.
         *
         * The inquiry data is allocated/refreshed by scsi_probe/scsi_slave
         * and freed by uninitchild (inquiry data is no longer freed by
         * scsi_unprobe/scsi_unslave).
         *
         * NOTE: Additional device identity information may be available
         * as properties of sd_dev.
         */
        struct scsi_inquiry     *sd_inq;

        /*
         * Place to point to an extended request sense buffer.
         * The target driver is responsible for managing this.
         */
        struct scsi_extended_sense      *sd_sense;

        /*
         * Target driver 'private' information. Typically a pointer to target
         * driver private ddi_soft_state(9F) information for the device.  This
         * information is typically established in target driver attach(9E),
         * and freed in the target driver detach(9E).
         *
         * LEGACY: For a scsi_device structure allocated by scsi_vhci during
         * online of a path, this was set by scsi_vhci to point to the
         * pathinfo node. Please use sd_pathinfo instead.
         */
        void                    *sd_private;

        /*
         * FMA capabilities of scsi_device.
         */
        int                     sd_fm_capable;

        /*
         * mdi_pathinfo_t pointer to pathinfo node for scsi_device structure
         * allocated by the scsi_vhci for transport to a specific pHCI path.
         */
        void                    *sd_pathinfo;

        /*
         * sd_uninit_prevent - Counter that prevents demotion of
         * DS_INITIALIZED node (esp loss of devi_addr) by causing
         * DDI_CTLOPS_UNINITCHILD failure - devi_ref will not protect
         * demotion of DS_INITIALIZED node.
         *
         * sd_tran_tgt_free_done - in some cases SCSA will call
         * tran_tgt_free(9E) independent of devinfo node state, this means
         * that uninitchild code should not call tran_tgt_free(9E).
         */
        unsigned                sd_uninit_prevent:16,
        int                     sd_uninit_prevent:16,
                                sd_tran_tgt_free_done:1,
                                sd_flags_pad:15;

        /*
         * The 'sd_tran_safe' field is a grotty hack that allows direct-access
         * (non-scsa) drivers (like chs, ata, and mlx - which all make cmdk
         * children) to *illegally* put their own vector in the scsi_address(9S)
         * 'a_hba_tran' field. When all the drivers that overwrite
         * 'a_hba_tran' are fixed, we can remove sd_tran_safe (and make
         * scsi_hba.c code trust that the 'sd_address.a_hba_tran' established
         * during initchild is still valid when uninitchild occurs).
         *
         * NOTE: This hack is also shows up in the DEVP_TO_TRAN implementation
         * in scsi_confsubr.c.
         *
         * NOTE: The 'sd_tran_safe' field is only referenced by SCSA framework
         * code, so always keeping it at the end of the scsi_device structure
         * (until it can be removed) is OK.  It use to be called 'sd_reserved'.
         */
        struct scsi_hba_tran    *sd_tran_safe;

#ifdef  SCSI_SIZE_CLEAN_VERIFY
        /*
         * Must be last: Building a driver with-and-without
         * -DSCSI_SIZE_CLEAN_VERIFY, and checking driver modules for
         * differences with a tools like 'wsdiff' allows a developer to verify
         * that their driver has no dependencies on scsi*(9S) size.
         */
        int                     _pad[8];
#endif  /* SCSI_SIZE_CLEAN_VERIFY */
};
```

______***unchanged_portion_omitted_***