man pages section 7: Device and Network Interfaces

Beta

Part No: 819–2254–33
February 2010

100217@23474

Contents

Preface

Both novice users and those familar with the SunOS operating system can use online man pages to obtain information about the system and its features. A man page is intended to answer concisely the question "What does it do?" The man pages in general comprise a reference manual. They are not intended to be a tutorial.

Overview

The following contains a brief description of each man page section and the information it references:

- Section 1 describes, in alphabetical order, commands available with the operating system.
- Section 1M describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes.
- Section 2 describes all of the system calls. Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value.
- Section 3 describes functions found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in Section 2.
- Section 4 outlines the formats of various files. The C structure declarations for the file formats are given where applicable.
- Section 5 contains miscellaneous documentation such as character-set tables.
- Section 7 describes various special files that refer to specific hardware peripherals and device drivers. STREAMS software drivers, modules and the STREAMS-generic set of system calls are also described.
- Section 9E describes the DDI (Device Driver Interface)/DKI (Driver/Kernel Interface), DDI-only, and DKI-only entry-point routines a developer can include in a device driver.
- Section 9F describes the kernel functions available for use by device drivers.
- Section 9S describes the data structures used by drivers to share information between the driver and the kernel.

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report,

there is no BUGS section. See the `intro` pages for more information and detail about each section, and `man(1)` for more information about man pages in general.

NAME — This section gives the names of the commands or functions documented, followed by a brief description of what they do.

SYNOPSIS — This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full path name is shown. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

- [ ] — Brackets. The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.
- . . . — Ellipses. Several values can be provided for the previous argument, or the previous argument can be specified multiple times, for example, "`filename...`".
- | — Separator. Only one of the arguments separated by this character can be specified at a time.
- { } — Braces. The options and/or arguments enclosed within braces are interdependent, such that everything enclosed must be treated as a unit.

PROTOCOL — This section occurs only in subsection 3R to indicate the protocol description file.

DESCRIPTION — This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES. Interactive commands, subcommands, requests, macros, and functions are described under USAGE.

IOCTL — This section appears on pages in Section 7 only. Only the device class that supplies appropriate parameters to the `ioctl(2)` system call is called `ioctl` and generates its own heading. `ioctl` calls for a specific device are listed alphabetically (on the man page for that specific device).

	`ioctl` calls are used for a particular class of devices all of which have an `io` ending, such as `mtio(7I)`.
OPTIONS	This secton lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.
OPERANDS	This section lists the command operands and describes how they affect the actions of the command.
OUTPUT	This section describes the output – standard output, standard error, or output files – generated by the command.
RETURN VALUES	If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or –1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.
ERRORS	On failure, most functions place an error code in the global variable `errno` indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each condition is described in a separate paragraph under the error code.
USAGE	This section lists special rules, features, and commands that require in-depth explanations. The subsections listed here are used to explain built-in functionality: Commands Modifiers Variables Expressions Input Grammar
EXAMPLES	This section provides examples of usage or of how to use a command or function. Wherever possible a complete

	example including command-line entry and machine response is shown. Whenever an example is given, the prompt is shown as `example%`, or if the user must be superuser, `example#`. Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS, and USAGE sections.
ENVIRONMENT VARIABLES	This section lists any environment variables that the command or function affects, followed by a brief description of the effect.
EXIT STATUS	This section lists the values the command returns to the calling program or shell and the conditions that cause these values to be returned. Usually, zero is returned for successful completion, and values other than zero for various error conditions.
FILES	This section lists all file names referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.
ATTRIBUTES	This section lists characteristics of commands, utilities, and device drivers by defining the attribute type and its corresponding value. See `attributes(5)` for more information.
SEE ALSO	This section lists references to other man pages, in-house documentation, and outside publications.
DIAGNOSTICS	This section lists diagnostic messages with a brief explanation of the condition causing the error.
WARNINGS	This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.
NOTES	This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is never covered here.
BUGS	This section describes known bugs and, wherever possible, suggests workarounds.

REFERENCE

Introduction

Name Intro – introduction to special files

Description This section describes various device and network interfaces available on the system. The types of interfaces described include character and block devices, STREAMS modules, network protocols, file systems, and ioctl requests for driver subsystems and classes.

This section contains the following major collections:

(7D) The system provides drivers for a variety of hardware devices, such as disk, magnetic tapes, serial communication lines, mice, and frame buffers, as well as virtual devices such as pseudo-terminals and windows.

This section describes special files that refer to specific hardware peripherals and device drivers. STREAMS device drivers are also described. Characteristics of both the hardware device and the corresponding device driver are discussed where applicable.

An application accesses a device through that device's special file. This section specifies the device special file to be used to access the device as well as application programming interface (API) information relevant to the use of the device driver.

All device special files are located under the `/devices` directory. The `/devices` directory hierarchy attempts to mirror the hierarchy of system busses, controllers, and devices configured on the system. Logical device names for special files in `/devices` are located under the `/dev` directory. Although not every special file under `/devices` will have a corresponding logical entry under `/dev`, whenever possible, an application should reference a device using the logical name for the device. Logical device names are listed in the `FILES` section of the page for the device in question.

This section also describes driver configuration where applicable. Many device drivers have a driver configuration file of the form *driver_name*`.conf` associated with them (see `driver.conf(4)`). The configuration information stored in the driver configuration file is used to configure the driver and the device. Driver configuration files are located in `/kernel/drv` and `/usr/kernel/drv`. Driver configuration files for platform dependent drivers are located in `/platform/`‘`uname -i`‘`/kernel/drv` where ‘`uname -i`‘ is the output of the `uname(1)` command with the `-i` option.

Some driver configuration files may contain user configurable properties. Changes in a driver's configuration file will not take effect until the system is rebooted or the driver has been removed and re-added (see `rem_drv(1M)` and `add_drv(1M)`).

(7FS) This section describes the programmatic interface for several file systems supported by SunOS.

(7I) This section describes ioctl requests which apply to a class of drivers or subsystems. For example, ioctl requests which apply to most tape devices are discussed in `mtio(7I)`. Ioctl requests relevant to only a specific device are described on the man page for that device. The page for the device in question should still be examined for exceptions to the ioctls listed in section 7I.

(7M) This section describes STREAMS modules. Note that STREAMS drivers are discussed in section 7D. `streamio(7I)` contains a list of ioctl requests used to manipulate STREAMS modules and interface with the STREAMS framework. Ioctl requests specific to a STREAMS module will be discussed on the man page for that module.

(7P) This section describes various network protocols available in SunOS.

SunOS supports both socket-based and STREAMS-based network communications. The Internet protocol family, described in `inet(7P)`, is the primary protocol family supported by SunOS, although the system can support a number of others. The raw interface provides low-level services, such as packet fragmentation and reassembly, routing, addressing, and basic transport for socket-based implementations. Facilities for communicating using an Internet-family protocol are generally accessed by specifying the `AF_INET` address family when binding a socket; see `socket(3SOCKET)` for details.

Major protocols in the Internet family include:

- The Internet Protocol (IP) itself, which supports the universal datagram format, as described in `ip(7P)`. This is the default protocol for `SOCK_RAW` type sockets within the `AF_INET` domain.
- The Transmission Control Protocol (TCP); see `tcp(7P)`. This is the default protocol for `SOCK_STREAM` type sockets.
- The User Datagram Protocol (UDP); see `udp(7P)`. This is the default protocol for `SOCK_DGRAM` type sockets.
- The Address Resolution Protocol (ARP); see `arp(7P)`.
- The Internet Control Message Protocol (ICMP); see `icmp(7P)`.

See Also `add_drv(1M)`, `rem_drv(1M)`, `Intro(3)`, `ioctl(2)`, `socket(3SOCKET)`, `driver.conf(4)`, `arp(7P)`, `icmp(7P)`, `inet(7P)`, `ip(7P)`, `mtio(7I)`, `st(7D)`, `streamio(7I)`, `tcp(7P)`, `udp(7P)`

System Administration Guide: IP Services

STREAMS Programming Guide

Writing Device Drivers

REFERENCE

Device and Network Interfaces

Name aac – SCSI HBA driver for Adaptec AdvancedRAID Controller

Description The aac plain SCSI host bus adapter driver is a SCSA-compliant nexus driver that supports the Adaptec 2200S/2120S SCSI RAID card, Dell PERC 3Di SCSI RAID controller, Dell PERC 3Si SCSI RAID controller, Adaptec 2820SA SATA RAID card, Adaptec 4800SAS, 4805SAS SAS RAID cards and SUN's STK RAID REM, STK RAID INT, and STK RAID EXT RAID cards.

The aac driver is ported from FreeBSD and supports RAID disk I/O functions and the RAID management interface.

Driver Configuration There are no user configurable parameters available. Please configure your hardware through BIOS.

Attributes See attributes(5) for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	x86, SPARC (Limited to systems with AAC hardware RAID cards.)

Files

/kernel/drv/aac	32-bit ELF kernel module.
/kernel/drv/amd64/aac	64-bit ELF kernel module. (x86)
/kernel/drv/sparcv9/aac	64-bit ELF kernel module. (SPARC)
/kernel/drv/aac.conf	Configuration file. (Contains no user-configurable options).

See Also prtconf(1M), attributes(5), scsi_hba_attach(9F), scsi_sync_pkt(9F), scsi_transport(9F), scsi_device(9S), scsi_inquiry(9S), scsi_pkt(9S)

Small Computer System Interface-2 (SCSI-2)

Name adp – Low-level module for controllers based on Adaptec AIC-7870P and AIC-7880P SCSI chips

Description The adp module provides low-level interface routines between the common disk/tape I/O system and SCSI (Small Computer System Interface) controllers based on the Adaptec AIC-7870P and AIC-7880P SCSI chips. These controllers include the Adaptec AHA–2940, AHA–2940W, AHA–2940U, AHA–2940UW, AHA–3940, and AHA–3940W, as well as motherboards with embedded AIC-7870P and AIC-7880P SCSI chips.

Supported devices are AIC-7850, AIC-7860, AIC-7870, AIC-7880 and AIC-7895.

The adp module can be configured for disk and streaming tape support for one or more host adapter boards, each of which must be the sole initiator on a SCSI bus. Auto-configuration code determines if the adapter is present at the configured address and what types of devices are attached to the adapter.

Preconfigure The Plug N Play SCAM Support option is not supported.

Known Problems and Limitations

- To use the AHA-3940 or AHA-3940W adapters, the motherboard must have a BIOS that supports the DEC PCI-to-PCI Bridge chip on the host bus adapter.
- User-level programs have exhibited problems on some PCI systems with an Adaptec AHA-2940 or AHA-2940W card and certain motherboards. If problems with user-level programs occur, use the BIOS setup to disable write-back CPU caching (or all caching if there is no control over the caching algorithm). The affected motherboards are:

- PCI motherboards with a 60-MHz Pentium chip, with PCI chipset numbers S82433LX Z852 and S82434LX Z850. The part numbers of the Intel motherboards are AA616393-007 and AA615988-009.
- PCI motherboards with a 90-MHz Pentium chip, with PCI chipset numbers S82433NX Z895, S82434NX Z895, and S82434NX Z896. The part number of the Intel motherboard is 541286-005. (Some Gateway 2000 systems use this motherboard.)
- AA-619772-002 motherboard with 82433LX Z852 and 82434LX Z882 chips causes random memory inconsistencies. Return the motherboard to the vendor for a replacement.

- If the AHA-2940 SCSI adapter does not recognize the Quantum Empire 1080S, HP 3323 SE or other SCSI disk drive, reduce the Synchronous Transfer rate on the Adaptec controller to 8 Mbps.
- The AHA-3940 has been certified by Adaptec to work on specific systems; however, some testing has shown that the Solaris operating environment works properly in some of those systems and not in others.

Configuration Use the Adaptec configuration utility to perform the following steps:

- Configure each SCSI device to have a unique SCSI ID, then using the adapter's Advanced Configuration Options setup menu, set the Plug N Play SCAM Support option to Disabled.
- If there is more than one controller (or an embedded controller), try to use one IRQ per controller.
- Enable bus mastering for the slots with your host bus adapters, when the choice is given.
- For older disk drives, tape drives, and most CD-ROM devices, make sure the maximum SCSI data transfer speed is set to 5.0 Mbps.
- Enable support for disks larger than 1 Gbyte if applicable.

Files

`/kernel/drv/adp.conf`	Configuration file for the `adp` driver; there are no user-configurable options in this file

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86

See Also

`attributes(5)`

Solaris (Intel Platform Edition) Hardware Compatibility List

Notes

Throughout the release, support of additional devices may be added. See the *Solaris (Intel Platform Edition) Hardware Compatibility List* for additional information.

The `adp` driver supports Logical Unit Number ("LUN") values of 0 through 15. This range exceeds the standard SCSI-2 requirements which call for support of LUNs 0 through 7.

Name adpu320 – Adaptec Ultra320 SCSI host bus adapter driver

Synopsis `scsi@unit-address`

Description The `adpu320` host bus adapter driver is a SCSA-compliant nexus driver that supports the following Adaptec Ultra320 SCSI Devices:

Chips AIC-7902

The `adpu320` driver supports standard functions provided by the SCSA interface, including tagged and untagged queuing, Wide/Fast/Ultra SCSI, and auto request sense. The `adpu320` driver does not support linked commands. The `adpu320` driver supports hot swap SCSI and hot plug PCI.

Additionally, the `adpu320` driver supports the following features:

- 64-bit addressing (Dual address Cycle)
- PCI-X v1.1 operating up to 133MHz and 64bits
- PCI bus spec v2.2 operating up to 66MHz and 64bits
- Packetized SCSI at 320 and 160 MB/s
- QAS
- DT
- 40MB/sec in single-ended mode and up to 320MB/sec transfer rate in LVD mode
- Domain Validation
- Retained Training Information (RTI)
- PCI and PCI-X Error handling

Note – The `adpu320` driver does not support the HostRAID feature found on some Adaptec SCSI controllers. For `adpu320` to support a Adaptec SCSI adapter with HostRAID, you must not use any HostRAID features.

Driver Configuration The `adpu320` host bus adapter driver is configured by defining the properties found in `adpu320`.conf. Properties in the `adpu320`.conf file that can be modified by the user include: `ADPU320_SCSI_RD_STRM`, `ADPU320_SCSI_NLUN_SUPPORT`.

```
-------------------------------------------------------------
               Option: ADPU320_SCSI_RD_STRM=[value]
           Definition: Enables/disables read streaming negotiation
                       for all drives.
      Possible Values: 0 (off), 1 (on)
        Default Value: 0 (off)

               Option: ADPU320_SCSI_NLUN_SUPPORT=[value]
           Definition: Enables the number of logical units to be
                       scanned per drive.
      Possible Values: 1-64
        Default Value: 64
    -------------------------------------------------------------
```

If you alter or add driver parameters incorrectly, you can render your system inoperable. Use driver parameters with caution.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	x86

Files

`/kernel/drv/adpu320`	Driver module.
`/kernel/drv/amd64/adpu320`	64–bit driver module.
`/kernel/drv/adpu320.conf`	Configuration file.

See Also `cfgadm(1M)`, `prtconf(1M)`, `attributes(5)`, `scsi_abort(9F)`, `scsi_hba_attach_setup(9F)`, `scsi_ifgetcap(9F)`, `scsi_reset(9F)`, `scsi_sync_pkt(9F)`, `scsi_transport(9F)`, `scsi_device(9S)`, `scsi_extended_sense(9S)`, `scsi_inquiry(9S)`, `scsi_pkt(9S)`

Writing Device Drivers

Small Computer System Interface-3 (SCSI-3)

Name afb – Elite3D graphics accelerator driver

Description The `afb` driver is the device driver for the Sun Elite3D graphics accelerators. The `afbdaemon`process loads the `afb` microcode at system startup time and during the resume sequence of a suspend-resume cycle.

Files

`/dev/fbs/afb`*n*	Device special file
`/usr/lib/afb.ucode`	afb microcode
`/usr/sbin/afbdaemon`	afb microcode loader

See Also `afbconfig(1M)`

Name afe – ADMtek Fast Ethernet device driver

Synopsis `/dev/afe`

Description The `afe` driver is a multi-threaded, loadable, clonable, GLD-based STREAMS driver supporting the Data Link Provider Interface `dlpi(7P)` on ADMtek (now Infineon) Centaur and Comet Fast Ethernet controllers.

Application Programming Interface The `afe` driver can be used as either a "style 1" or a "style 2" Data Link Service Provider. Physical points of attachment (PPAs) are interpreted as the instance number of the `afe` controller as assigned by the Solaris operating environment.

The relevant fields returned as part of a `DL_INFO_ACK` response are:

- Maximum SDU is 1500.
- Minimum SDU is 0.
- The dlsap address length is 8.
- MAC type is `DL_ETHER`.
- SAP length is -2. The 6–byte physical address is followed immediately by a 2–byte SAP.
- Service mode is `DL_CLDLS`.
- Broadcast address is the 6-byte Ethernet broadcast address (`ff:ff:ff:ff:ff:ff`).

If the SAP provided is zero, then *IEEE 802.3* mode is assumed and outbound frames will have the frame payload length written into the type field. Likewise, inbound frames with a SAP between zero and 1500 are interpreted as *IEEE 802.3* frames and delivered to any streams that are bound to SAP zero (the *802.3* SAP).

Properties The following properties may be configured using either `ndd(1M)` or the `afe.conf` configuration file as described by `driver.conf(4)`:

`adv_autoneg_cap`
Enables (default) or disables *IEEE 802.3u* auto-negotiation of link speed and duplex settings. If enabled, the device negotiates among the supported (and configured, see below) link options with the link partner. If disabled, at least one of the link options below must be specified. The driver selects the first enabled link option according to the *IEEE 802.3u* specified preferences.

`adv_100T4_cap`
Enables the 100 BaseT4 link option. (Note that most hardware does not support this unusual link style. Also, this uses two pairs of wires for data, rather than one.)

`adv_100fdx_cap`
Enables the 100 Base TX full-duplex link option. (This is generally the fastest mode if both link partners support it. Most modern equipment supports this mode.)

`adv_100hdx_cap`
Enables the 100 Base TX half-duplex link option. (Typically used when the link partner is a 100 Mbps hub.)

`adv_10fdx_cap`
Enables the 10 Base-T full-duplex link option. (This less-frequently used mode is typically used when the link partner is a 10 Mbps switch.)

`adv_10hdx_cap`
Enables the 10 Base-T half-duplex link option. (This is the fall-back when no other option is available. It is typically used when the link partner is a 10 Mbps hub or is an older network card.)

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SPARC, x86
Interface Stability	Committed

Files `/dev/afe`
Special character device.

`/kernel/drv/afe`
32-bit driver binary (x86).

`/kernel/drv/amd64/afe`
64-bit driver binary (x86).

`/kernel/drv/sparcv9/afe`
64-bit driver binary (SPARC).

`/kernel/drv/afe.conf`
Configuration file.

See Also `ndd(1M)`, `driver.conf(4)`, `attributes(5)`, `streamio(7I)`, `dlpi(7P)`

IEEE 802.3 — Institute of Electrical and Electronics Engineers, 2002

Name agpgart_io – Solaris agpgart driver I/O control operations

Synopsis

```
#include <sys/agpgart.h>
```

Description The Accelerated Graphics Port (AGP) is a PCI bus technology enhancement that improves 3D graphics performance by using low-cost system memory. AGP chipsets use the Graphics Address Remapping Table (GART) to map discontiguous system memory into a contiguous PCI memory range (known as the AGP Aperture), enabling the graphics card to utilize the mapped aperture range as video memory.

The `agpgart` driver creates a pseudo device node at `/dev/agpgart` and provides a set of ioctls for managing allocation/deallocation of system memory, setting mappings between system memory and aperture range, and setting up AGP devices. The `agpgart` driver manages both pseudo and real device nodes, but to initiate AGP-related operations you operate only on the `/dev/agpgart` pseudo device node. To do this, open `/dev/agpgart`. The macro defined for the pseudo device node name is:

```
#define  AGP_DEVICE      "/dev/agpgart"
```

The `agpgart_io` driver implementation is AGP architecture-dependent and cannot be made generic. Currently, the `agpgart_io` driver only supports specific AGP systems. To determine if a system is supported, run an `open(2)` system call on the AGP_DEVICE node. (Note that `open(2)` fails if a system is not supported). After the AGP_DEVICE is opened, you can use `kstat(1M)` to read the system architecture type.

In addition to AGP system support, the `agpgart` ioctls can also be used on Intel integrated graphics devices (IGD). IGD devices usually have no dedicated video memory and must use system memory as video memory. IGD devices contain translation tables (referred to as *GTT* tables) that are similar to the GART translation table for address mapping purposes.

Processes must open the `agpgart_io` driver utilizing a GRAPHICS_ACCESS privilege. Then all the ioctls can be called by this processes with the saved file descriptor. With the exception of AGPIOC_INFO, the AGPIOC_ACQUIRE ioctl must be called before any other ioctl. Once a process has acquired GART, it cannot be acquired by another process until the former process calls AGPIOC_RELEASE.

If the AGP_DEVICE fails to open, it may be due to one of the following reasons:

EAGAIN
: GART table allocation failed.

EIO
: Internal hardware initialization failed.

ENXIO
: Getting device soft state error. (This is unlikely to happen.)

EPERM
: Without enough privilege.

ioctls With the exception of GPIOC_INFO, all ioctls shown in this section are protected by GRAPHICS_ACCESS privilege. (Only processes with GRAPHICS_ACCESS privilege in its effective set can access the privileged ioctls).

Common ioctl error codes are shown below. (Additional error codes may be displayed by individual ioctls.)

ENXIO
Ioctl command not supported or getting device soft state error.

EPERM
Process not privileged.

`AGPIOC_INFO`
Get system wide AGP or IGD hardware information. This command can be called by any process from user or kernel context.

```
The argument is a pointer to agp_info_t structure.

 typedef struct  _agp_info {
       agp_version_t agpi_version; /* OUT: AGP version supported */
       uint32_t agpi_devid;    /* OUT: bridge vendor + device */
       uint32_t agpi_mode;     /* OUT: mode of bridge */
       ulong_t  agpi_aperbase; /* OUT: base of aperture */
       size_t   agpi_apersize; /* OUT: aperture size in MB */
       uint32_t agpi_pgtotal;  /* OUT: max aperture pages avail. */
       uint32_t agpi_pgsystem; /* OUT: same as pg_total */
       uint32_t agpi_pgused; /* OUT: no. of currently used pages */
 } agp_info_t;

agpi_version  The version of AGP protocol the bridge device is
              compatible with, for example, major 3 and minor 0
              means AGP version 3.0.

              typedef struct _agp_version {
                      uint16_t   agpv_major;
                      uint16_t   agpv_minor;
              } agp_version_t;

agpi_devid    AGP bridge vendor and device ID.
agpi_mode     Current AGP mode, read from AGP status register of
              target device. The main bits are defined as below.
              /* AGP status register bits definition */

                #define AGPSTAT_RQ_MASK         0xff000000
                #define AGPSTAT_SBA             (0x1 << 9)
                #define AGPSTAT_OVER4G          (0x1 << 5)
                #define AGPSTAT_FW              (0x1 << 4)
                #define AGPSTAT_RATE_MASK       0x7
```

```
                    /* AGP 3.0 only bits */
                    #define AGPSTAT_ARQSZ_MASK      (0x7 << 13)
                    #define AGPSTAT_CAL_MASK        (0x7 << 10)
                    #define AGPSTAT_GART64B         (0x1 << 7)
                    #define AGPSTAT_MODE3           (0x1 << 3)
                    /* rate for 2.0 mode */
                    #define AGP2_RATE_1X                    0x1
                    #define AGP2_RATE_2X                    0x2
                    #define AGP2_RATE_4X                    0x4
                    /* rate for 3.0 mode */
                    #define AGP3_RATE_4X                    0x1
                    #define AGP3_RATE_8X                    0x2

  agpi_aperbase   The base address of aperture in PCI memory space.
  agpi_apersize   The size of the aperture in megabytes.
  agpi_pgtotal    Represents the maximum memory
                  pages the system can allocate
                  according to aperture size and
                  system memory size (which may differ
                  from the maximum locked memory a process
                  can have. The latter  is subject
                  to the memory resource limit imposed
                  by the  resource_controls(5) for each
                  project(4)):

                    project.max-device-locked-memory

                  This value can be modified through system
                  utilities like prctl(1).

  agpi_pgsystem  Same as pg_total.
  agpi_pgused    System pages already allocated by the driver.

  Return Values:

             EFAULT  Argument copy out error
             EINVAL  Command invalid
             0       Success
```

AGPIOC_ACQUIRE

Acquire control of GART. With the exception of AGPIOC_INFO, a process must acquire GART before can it call other agpgart ioctl commands. Additionally, only processes with GRAPHICS_ACCESS privilege may access this ioctl. In the current agpgart implementation, GART access is exclusive, meaning that only one process can perform GART operations at a time. To release control over GART, call AGPIOC_RELEASE. This command can be called from user or kernel context.

The argument should be NULL.

Return values:

EBUSY GART has been acquired

0 Success.

AGPIOC_RELEASE

Release GART control. If a process releases GART control, it cannot perform additional GART operations until GART is reacquired. Note that this command does not free allocated memory or clear GART entries. (All clear jobs are done by direct calls or by closing the device). When a process exits without making this ioctl, the final close(2) performs this automatically. This command can be called from user or kernel context.

The argument should be NULL.

Return values:

EPERM Not owner of GART.

0 Success.

AGPIOC_SETUP

Setup AGPCMD register. An AGPCMD register resides in both the AGP master and target devices. The AGPCMD register controls the working mode of the AGP master and target devices. Each device must be configured using the same mode. This command can be called from user or kernel context.

```
The argument is a pointer to agp_setup_t structure:

     typedef struct _agp_setup {
           uint32_t agps_mode; /* IN: value to be set for AGPCMD */
     } agp_setup_t;

agps_mode  Specifying the mode to be set. Each bit of the value may have
           a specific meaning, please refer to AGP 2.0/3.0 specification
           or hardware datasheets for details.

               /* AGP command register bits definition */
               #define     AGPCMD_RQ_MASK          0xff000000
               #define     AGPCMD_SBAEN            (0x1 << 9)
               #define     AGPCMD_AGPEN            (0x1 << 8)
               #define     AGPCMD_OVER4GEN         (0x1 << 5)
               #define     AGPCMD_FWEN             (0x1 << 4)
               #define     AGPCMD_RATE_MASK        0x7
               /* AGP 3.0 only bits */
               #define     AGP3_CMD_ARQSZ_MASK     (0x7 << 13)
               #define     AGP3_CMD_CAL_MASK       (0x7 << 10)
               #define     AGP3_CMD_GART64BEN      (0x1 << 7)
```

The final values set to the AGPCMD register of the master/target devices are decided by the agps_mode value and AGPSTAT of the master and target devices.

Return Values:

EPERM	Not owner of GART.
EFAULT	Argument copy in error.
EINVAL	Command invalid for non-AGP system.
EIO	Hardware setup error.
0	Success.

AGPIOC_ALLOCATE

Allocate system memory for graphics device. This command returns a unique ID which can be used in subsequent operations to represent the allocated memory. The memory is made up of discontiguous physical pages. In rare cases, special memory types may be required. The allocated memory must be bound to the GART table before it can be used by graphics device. Graphics applications can also mmap(2) the memory to userland for data storing. Memory should be freed when it is no longer used by calling AGPIOC_DEALLOCATE or simply by closing the device. This command can be called from user or kernel context.

The argument is a pointer to agp_allocate_t structure.

```
typedef struct _agp_allocate {
      int32_t  agpa_key;     /* OUT:ID of allocated memory */
      uint32_t agpa_pgcount;/* IN: no. of pages to be allocated */
      uint32_t agpa_type;/* IN: type of memory to be allocated */
      uint32_t agpa_physical; /* OUT: reserved */
} agp_allocate_t;
```

agpa_key	Unique ID of the allocated memory.
agpa_pgcount	Number of pages to be allocated. The driver currently supports only 4K pages. The value cannot exceed the agpi_pgtotal value returned by AGPIOC_INFO ioct and is subject to the limit of project.max-device-locked-memory. If the memory needed is larger than the resource limit but not larger than agpi_pgtotal, use prctl(1) or other system utilities to change the default value of memory resource limit beforehand.
agpa_type	Type of memory to be allocated. The valid value of agpa_type should be AGP_NORMAL. It is defined as: `#define AGP_NORMAL 0` Above, AGP_NORMAL represents the discontiguous non-cachable physical memory which doesn't consume

kernel virtual space but can be mapped to user space by mmap(2). This command may support more type values in the future.

agpa_physical — Reserved for special uses. In normal operations, the value is undefined.

Return Values:

EPERM	Not owner of GART.
EINVAL	Argument not valid.
EFAULT	Argument copy in/out error.
ENOMEM	Memory allocation error.
0	Success.

AGPIOC_DEALLOCATE — Deallocate the memory identified by a key assigned in a previous allocation. If the memory isn't unbound from GART, this command unbinds it automatically. The memory should no longer be used and those still in mapping to userland cannot be deallocated. Always call AGPIOC_DEALLOCATE explicitly (instead of deallocating implicitly by closing the device), as the system won't carry out the job until the last reference to the device file is dropped. This command from user or kernel context.

The input argument is a key of type int32_t, no output argument.

Return Values:

EPERM	Not owner of GART.
EINVAL	Key not valid or memory in use.
0	Success.

AGPIOC_BIND — Bind allocated memory. This command binds the allocated memory identified by a key to a specific offset of the GART table, which enables GART to translate the aperture range at the offset to system memory. Each GART entry represents one physical page. If the GART range is previously bound to other system memory, it returns an error. Once the memory is bound, it cannot be bound to other offsets unless it is unbound. To unbind the memory, call AGPIOC_UNBIND or deallocate the memory. This command can be called from user or kernel context.

The argument is a pointer to agp_bind_t structure:

```
typedef struct _agp_bind {
      int32_t  agpb_key;      /* IN: ID of memory to be bound */
      uint32_t agpb_pgstart;  /* IN: offset in aperture */
} agp_bind_t;
```

	agpb_key	The unique ID of the memory to be bound, which is previously allocated by calling AGPIOC_ALLOCATE.
	agpb_pgstart	The starting page offset to be bound in aperture space.
	Return Values:	
	EPERM	Not owner of GART.
	EFAULT	Argument copy in error.
	EINVAL	Argument not valid.
	EIO	Binding to the GTT table of IGD devices failed.
	0	Success.
AGPIOC_UNBIND	Unbind memory identified by a key from the GART. This command clears the corresponding entries in the GART table. Only the memory not in mapping to userland is allowed to be unbound. This ioctl command can be called from user or kernel context.	

The argument is a pointer to agp_unbind_t structure.

```
typedef struct _agp_unbind {
      int32_t  agpu_key; /* IN: key of memory to be unbound*/
      uint32_t agpu_pri; /* Not used: for compat. with Xorg */
} agp_unbind_t;
```

	agpu_key	Unique ID of the memory to be unbound which was previously bound by calling AGPIOC_BIND.
	agpu_pri	Reserved for compatibility with X.org/XFree86, not used.
	Return Values:	

EPERM	Not owner of GART.
EFAULT	Argument copy in error.
EINVAL	Argument not valid or memory in use.
EIO	Unbinding from the GTT table of IGD devices failed.
0	Success

Example Below is an sample program showing how agpgart ioctls can be used:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h
#include <sys/ioccom.h>
#include <sys/types.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/mman.h>
#include <sys/agpgart.h>

#define AGP_PAGE_SIZE   4096

int main(int argc, char *argv[])
{
        int fd, ret;
        agp_allocate_t alloc;
        agp_bind_t bindinfo;
        agp_info_t agpinfo;
        agp_setup_t modesetup;
        int *p = NULL;
        off_t mapoff;
        size_t maplen;

        if((fd = open(AGP_DEVICE, O_RDWR))== -1) {
                printf("open AGP_DEVICE error with %d\n", errno);\
                exit(-1);
        }
        printf("device opened\n");

        ret = ioctl(fd, AGPIOC_INFO, &agpinfo);
        if(ret == -1) {
                printf("Get info error %d\n", errno);
                exit(-1);
        }
        printf("AGPSTAT is %x\n", agpinfo.agpi_mode);
        printf("APBASE is %x\n", agpinfo.agpi_aperbase);
```

```
printf("APSIZE is %dMB\n", agpinfo.agpi_apersize);
printf("pg_total is %d\n", agpinfo.agpi_pgtotal);

ret = ioctl(fd, AGPIOC_ACQUIRE);
if(ret == -1) {
        printf(" Acquire GART error %d\n", errno);
        exit(-1);
}

modesetup.agps_mode = agpinfo.agpi_mode;
ret = ioctl(fd, AGPIOC_SETUP, &modesetup);
if(ret == -1) {
        printf("set up AGP mode error\n", errno);
        exit(-1);
}

printf("Please input the number of pages you want to allocate\n");
scanf("%d", &alloc.agpa_pgcount);
alloc.agpa_type = AGP_NORMAL;
ret = ioctl(fd, AGPIOC_ALLOCATE, &alloc);
if(ret == -1) {
        printf("Allocate memory error %d\n", errno);
        exit(-1);
}

printf("Please input the aperture page offset to bind\n");
scanf("%d", &bindinfo.agpb_pgstart);
bindinfo.agpb_key = alloc.agpa_key;
ret = ioctl(fd, AGPIOC_BIND, &bindinfo);
if(ret == -1) {
        printf("Bind error %d\n", errno);
        exit(-1);
}
printf("Bind successful\n");

/*
 * Now gart aperture space from (bindinfo.agpb_pgstart) to
 * (bindinfo.agpb_pgstart + alloc.agpa_pgcount) can be used for
 * AGP graphics transactions
 */
...

/*
 * mmap can allow user processes to store graphics data
 * to the aperture space
 */
maplen = alloc.agpa_pgcount * AGP_PAGE_SIZE;
```

```
        mapoff = bindinfo.agpb_pgstart * AGP_PAGE_SIZE;
        p = (int *)mmap((caddr_t)0, maplen, (PROT_READ | PROT_WRITE),
                        MAP_SHARED, fd, mapoff);
        if (p == MAP_FAILED) {
                printf("Mmap error %d\n", errno);
                exit(-1);
        }
        printf("Mmap successful\n");
        ...

        /*
         * When user processes finish access to the aperture space,
         * unmap the memory range
         */
        munmap((void *)p, maplen);
        ...

        /*
         * After finishing AGP transactions, the resources can be freed
         * step by step or simply by close device.
         */
        ret = ioctl(fd, AGPIOC_DEALLOCATE, alloc.agpa_key);
        if(ret == -1) {
                printf(" Deallocate memory error %d\n", errno);
                exit(-1);
        }

        ret = ioctl(fd, AGPIOC_RELEASE);
        if(ret == -1) {
                printf(" Release GART error %d\n", errno);
                exit(-1);
        }

        close(fd);
}
```

Files `/dev/agpgart`
Symbolic link to the pseudo agpgart device.

`/platform/i86pc/kernel/drv/agpgart`
agpgart pseudo driver.

`/platform/i86pc/kernel/drv/agpgart.conf`
Driver configuration file.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	X86
Availability	SUNWagp, SUNWagph
Stability level	Unstable

See Also prctl(1), kstat(1M), close(2), ioctl(2), open(2), mmap(2), project(4), privileges(5), attributes(5), resource_controls(5)

Name ahci – Advanced Host Controller Interface SATA controller driver

Synopsis `sata@unit-address`

Description The `ahci` driver is a SATA framework-compliant HBA driver that supports SATA HBA controllers that are compatible with the *Advanced Host Controller Interface 1.0* specification. AHCI is an Intel-developed protocol that describes the register-level interface for host controllers for serial ATA 1.0a and Serial ATA II. The *AHCI 1.0* specification describes the interface between the system software and the host controller hardware.

The `ahci` driver currently supports the Intel ICH6/7/8/9/10, VIA vt8251 and JMicron AHCI controllers which are compliant with the Advanced Host Controller Interface 1.0 specification. The Intel ICH6/7/8/9 and VIA vt8251 controllers support standard SATA features. The `ahci` driver currently supports hard disk, ATAPI DVD, ATAPI tape, ATAPI disk (i.e. Dell RD1000), hotplug, NCQ (Native command queuing) and Port multipliers (Silicon Image 3726/4726). Power management is not yet supported.

Configuration The `ahci` module contains no user configurable parameters.

Files `/kernel/drv/ahci`
32–bit ELF kernel module (x86).

`/kernel/drv/amd64/ahci`
64–bit ELF kernel module (x86).

Attributes See `attributes(5)` for descriptions of the following attribute:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWahci

See Also `cfgadm(1M)`, `cfgadm_sata(1M)`, `prtconf(1M)`, `sata(7D)`

Advanced Host Controller Interface 1.0

Writing Device Drivers

Notes To bind the `ahci` driver to your controller, choose the [AHCI] BIOS option.

Note that booting is not supported if toggle exists between the [IDE] and [AHCI] BIOS options

Name amd8111s – AMD-8111 Fast Ethernet Network Adapter driver

Synopsis `/dev/amd8111s`

Description The `amd8111s` Fast Ethernet driver is a multi-threaded, loadable, clonable, GLD-based STREAMS driver supporting the Data Link Provider Interface, `dlpi(7P)`, on the AMD-8111 Fast Ethernet Network Adapter.

The `amd8111s` driver functions include controller initialization, frame transmit and receive, promiscuous and multicast support, and error recovery and reporting.

Application Programming Interface The cloning, character-special device `/dev/amd8111` is used to access all AMD-8111 Fast Ethernet devices installed within the system.

The `amd8111s` driver is managed by the `dladm(1M)` command line utility, which allows VLANs to be defined on top of amd8111s instances and for amd8111s instances to be aggregated. See `dladm(1M)` for more details.

Configuration By default, the `amd8111s` driver performs auto-negotiation to select the link speed and mode. Link speed and mode can be any of the following:

100 Mbps, full-duplex.

100 Mbps, half-duplex.

10 Mbps, full-duplex.

10 Mbps, half-duplex.

Files

`/dev/amd8111s*`	Special character device.
`/kernel/drv/amd8111s*`	32-bit ELF kernel module (x86).
`/kernel/drv/amd64/amd8111s*`	64-bit ELF Kernel module (x86).

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNW8111s
Architecture	x86
Interface Stability	Committed

See Also `dladm(1M)`, `attributes(5)`, `streamio(7I)`, `dlpi(7P)`

Writing Device Drivers

STREAMS Programming Guide

Network Interfaces Programmer's Guide

Name amr – SCSI HBA driver for Dell PERC 3/DC, 4/SC, 4/DC and 4/DI

Description The `amr` plain SCSI host bus adapter driver is a SCSA-compliant nexus driver that supports the Dell PERC 3DC/4SC/4DC/4Di RAID devices.

The `amr` driver ports from FreeBSD and only supports basic RAID disk I/O functions.

Driver Configuration There are no user configurable parameters available. Please configure your hardware through BIOS.

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	x86

Files

`/kernel/drv/amr`	32-bit ELF kernel module.
`/kernel/drv/amd64/amr`	64–bit kernel module (x86 only).
`/kernel/drv/amr.conf`	Driver configuration file (contains no user-configurable options).

See Also `prtconf(1M)`, `attributes(5)`, `lsimega(7D)`, `scsi_hba_attach_setup(9F)`, `scsi_sync_pkt(9F)`, `scsi_transport(9F)`, `scsi_inquiry(9S)`, `scsi_device(9S)`, `scsi_pkt(9S)`

Small Computer System Interface-2 (SCSI-2)

Name arcmsr – SAS and SATA HBA driver for Areca Hardware Raid devices

Description The `arcmsr` host bus adapter driver is a SCSA-compliant nexus driver that supports Areca Technology SAS and SATA RAID devices.

Supported SATA RAID cards:

```
ARC-1110  pci17d3,1110
ARC-1120  pci17d3,1120
ARC-1130  pci17d3,1130
ARC-1160  pci17d3,1160
ARC-1170  pci17d3,1170
ARC-1201  pci17d3,1201
ARC-1210  pci17d3,1210
ARC-1220  pci17d3,1220
ARC-1230  pci17d3,1230
ARC-1260  pci17d3,1260
ARC-1270  pci17d3,1270
ARC-1280  pci17d3,1280
```

Supported SAS RAID cards:

```
ARC-1380  pci17d3,1380
ARC-1381  pci17d3,1381
ARC-1680  pci17d3,1680
ARC-1681  pci17d3,1681
```

Configuration There are no user configurable parameters available. Please configure your hardware through the host system BIOS.

Files

/kernel/drv/arcmsr	32-bit ELF kernel module.
/kernel/drv/amd64/arcmsr	64-bit kernel module (x64 only).
/kernel/drv/arcmsr.conf	Driver configuration file (contains no user-configurable options).

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86, x64 only
Availability	SUNWarcmsr

See Also `prtconf(1M)`, `attributes(5)`, `scsi_hba_attach_setup(9F)`, `scsi_sync_pkt(9F)`, `scsi_transport(9F)`, `scsi_inquiry(9S)`, `scsi_device(9S)`, `scsi_pkt(9S)`

Small Computer System Interface-2 (SCSI-2)

http://www.areca.com.tw/products/main.htm

http://developer.intel.com/design/iio/index.htm —(Intel Corp IO processors provide the underlying RAID engine for the supported devices).

Name arn – Atheros AR9280/9281/9285 IEEE802.11 a/b/g/n wireless network device

Description The `arn` IEEE802.11 a/b/g/n wireless driver is a loadable, clonable, GLDv3-based STREAMS driver supporting Atheros AR9280/9281/9285 IEEE802.11 a/b/g/n wireless network device.

Configuration The `arn` driver performs auto-negotiation to determine the data rate and mode. The driver supports only BSS networks (also known as `ap` or `infrastructure` networks) and `open`(or `open-system`) or `shared system` authentication. For wireless security, WEP encryption, WPA-PSK, and WPA2-PSK are currently supported. Configuration and administration tasks can be performed with the `dladm`(1M) and `wificonfig`(1M) utilities.

Files

`/dev/arn`	Special character device
`/kernel/drv/arn`	32-bit ELF kernel module (x86)
`/kernel/drv/amd64/arn`	64-bit ELF kernel driver module (x86)

Attributes See `attributes`(5) for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWarn
Interface Stability	Committed

See Also `dladm`(1M), `wificonfig`(1M), `attributes`(5), `dlpi`(7P), `gld`(7D)

Name arp, ARP – Address Resolution Protocol

Synopsis

```
#include <sys/fcntl.h>

#include <sys/socket.h>

#include <net/if_arp.h>

#include <netinet/in.h>

s = socket(AF_INET, SOCK_DGRAM, 0);

d = open ("/dev/arp", oflag);
```

Description ARP is a protocol used to map dynamically between Internet Protocol (IP) and Ethernet addresses. It is used by all Ethernet datalink providers (network drivers) and can be used by other datalink providers that support broadcast, including FDDI and Token Ring. The only network layer supported in this implementation is the Internet Protocol, although ARP is not specific to that protocol.

ARP caches IP-to-link-layer address mappings. When an interface requests a mapping for an address not in the cache, ARP queues the message that requires the mapping and broadcasts a message on the associated network requesting the address mapping. If a response is provided, ARP caches the new mapping and transmits any pending message. ARP will queue a maximum of four packets while awaiting a response to a mapping request. ARP keeps only the first four transmitted packets.

Application Programming Interface The STREAMS device `/dev/arp` is not a Transport Level Interface (TLI) transport provider and may not be used with the TLI interface.

To facilitate communications with systems that do not use ARP, ioctl() requests are provided to enter and delete entries in the IP-to-link address tables. Ioctls that change the table contents require sys_net_config privilege. See `privileges(5)`.

```
#include <sys/sockio.h>
#include <sys/socket.h>
#include <net/if.h>
#include <net/if_arp.h>
struct arpreq arpreq;
ioctl(s, SIOCSARP, (caddr_t)&arpreq);
ioctl(s, SIOCGARP, (caddr_t)&arpreq);
ioctl(s, SIOCDARP, (caddr_t)&arpreq);
```

`SIOCSARP`, `SIOCGARP` and `SIOCDARP` are BSD compatible ioctls. These ioctls do not communicate the mac address length between the user and the kernel (and thus only work for 6 byte wide Ethernet addresses). To manage the ARP cache for media that has different sized mac addresses, use `SIOCSXARP`, `SIOCGXARP` and `SIOCDXARP` ioctls.

```
#include <sys/sockio.h>
#include <sys/socket.h>
#include <net/if.h>
```

```
#include <net/if_dl.h>
#include <net/if_arp.h>
struct xarpreq xarpreq;
ioctl(s, SIOCSXARP, (caddr_t)&xarpreq);
ioctl(s, SIOCGXARP, (caddr_t)&xarpreq);
ioctl(s, SIOCDXARP, (caddr_t)&xarpreq);
```

Each `ioctl()` request takes the same structure as an argument. `SIOCS[X]ARP` sets an ARP entry, `SIOCG[X]ARP` gets an ARP entry, and `SIOCD[X]ARP` deletes an ARP entry. These `ioctl()` requests may be applied to any Internet family socket descriptors, or to a descriptor for the ARP device. Note that `SIOCS[X]ARP` and `SIOCD[X]ARP` require a privileged user, while `SIOCG[X]ARP`

does not.

The `arpreq` structure contains

```
/*
* ARP ioctl request
*/
struct arpreq {
    struct sockaddr arp_pa;  /* protocol address */
    struct sockaddr arp_ha; /* hardware address */
    int  arp_flags;         /* flags */
};
```

The `xarpreq` structure contains:

```
/*
* Extended ARP ioctl request
*/
struct xarpreq {
    struct sockaddr_storage xarp_pa; /* protocol address */
    struct sockaddr_dl xarp_ha;     /* hardware address */
    int xarp_flags;                 /* arp_flags field values */
};
#define ATF_COM 0x2          /* completed entry (arp_ha valid) */
#define ATF_PERM 0x4         /* permanent (non-aging) entry */
#define ATF_PUBL 0x8         /* publish (respond for other host) */
#define ATF_USETRAILERS 0x10 /* send trailer pckts to host */
#define ATF_AUTHORITY 0x20   /* hardware address is authoritative */
```

The address family for the [x]arp_pa sockaddr must be `AF_INET`. The `ATF_COM` flag bits ([x]arp_flags) cannot be altered. `ATF_USETRAILERS` is not implemented on Solaris and is retained for compatibility only. `ATF_PERM` makes the entry permanent (disables aging) if the `ioctl()` request succeeds. `ATF_PUBL` specifies that the system should respond to ARP requests for the indicated protocol address coming from other machines. This allows a host to act as an ARP server, which may be useful in convincing an ARP-only machine to talk to a non-ARP machine. `ATF_AUTHORITY` indicates that this machine owns the address. ARP does not update the entry based on received packets.

The address family for the arp_ha sockaddr must be AF_UNSPEC.

Before invoking any of the SIOC*XARP ioctls, user code must fill in the xarp_pa field with the protocol (IP) address information, similar to the BSD variant. The SIOC*XARP ioctls come in two (legal) varieties, depending on xarp_ha.sdl_nlen:

1. if sdl_nlen = 0, it behaves as an extended BSD ioctl. The kernel uses the IP address to determine the network interface.
2. if (sdl_nlen > 0) and (sdl_nlen < LIFNAMSIZ), the kernel uses the interface name in sdl_data[0] to determine the network interface; sdl_nlen represents the length of the string (excluding terminating null character).
3. if (sdl_nlen >= LIFNAMSIZ), an error (EINVAL) is flagged from the ioctl.

Other than the above, the xarp_ha structure should be 0-filled except for SIOCSXARP, where the sdl_alen field must be set to the size of hardware address length and the hardware address itself must be placed in the LLADDR/sdl_data[] area. (EINVAL will be returned if user specified sdl_alen does not match the address length of the identified interface).

On return from the kernel on a SIOCGXARP ioctl, the kernel fills in the name of the interface (excluding terminating NULL) and its hardware address, one after another, in the sdl_data/LLADDR area; if the two are larger than can be held in the 244 byte sdl_data[] area, an ENOSPC error is returned. Assuming it fits, the kernel will also set sdl_alen with the length of hardware address, sdl_nlen with the length of name of the interface (excluding terminating NULL), sdl_type with an IFT_* value to indicate the type of the media, sdl_slen with 0, sdl_family with AF_LINK and sdl_index (which if not 0) with system given index for the interface. The information returned is very similar to that returned via routing sockets on an RTM_IFINFO message.

The ARP ioctls have several additional restrictions and enhancements when used in conjunction with IPMP:

- ARP mappings for IPMP data and test addresses are managed by the kernel and cannot be changed through ARP ioctls, though they may be retrieved using SIOCGARP or SIOCGXARP.
- ARP mappings for a given IPMP group must be consistent across the group. As a result, ARP mappings cannot be associated with individual underlying IP interfaces in an IPMP group and must instead be associated with the corresponding IPMP IP interface.
- roxy ARP mappings for an IPMP group are automatically managed by the kernel. Specifically, if the hardware address in a SIOCSARP or SIOCSXARP request matches the hardware address of an IP interface in an IPMP group and the IP address is not local to the system, the kernel regards this as a IPMP Proxy ARP entry. This IPMP Proxy ARP entry will have its hardware address automatically adjusted in order to keep the IP address reachable (provided the IPMP group has not entirely failed).

—
—

—P

ARP performs duplicate address detection for local addresses. When a logical interface is brought up (`IFF_UP`) or any time the hardware link goes up (`IFF_RUNNING`), ARP sends probes (ar$spa == 0) for the assigned address. If a conflict is found, the interface is torn down. See `ifconfig(1M)` for more details.

ARP watches for hosts impersonating the local host, that is, any host that responds to an ARP request for the local host's address, and any address for which the local host is an authority. ARP defends local addresses and logs those with `ATF_AUTHORITY` set, and can tear down local addresses on an excess of conflicts.

ARP also handles UNARP messages received from other nodes. It does not generate these messages.

Packet Events

The `arp` driver registers itself with the netinfo interface. To gain access to these events, a handle from net_protocol_lookup must be acquired by passing it the value `NHF_ARP`. Through this interface, two packet events are supported:

Physical in - ARP packets received via a network inter face

Physical out - ARP packets to be sent out via a network interface

For ARP packets, the hook_pkt_event structure is filled out as follows:

hpe_ifp
: Identifier indicating the inbound interface for packets received with the `physical in` event.

hpe_ofp
: Identifier indicating the outbound interface for packets received with the `physical out` event.

hpe_hdr
: Pointer to the start of the ARP header (not the ethernet header).

hpe_mp
: Pointer to the start of the mblk_t chain containing the ARP packet.

hpe_mb
: Pointer to the mblk_t with the ARP header in it.

Network Interface Events

In addition to events describing packets as they move through the system, it is also possible to receive notification of events relating to network interfaces. These events are all reported back through the same callback. The list of events is as follows:

plumb
: A new network interface has been instantiated.

unplumb
A network interface is no longer associated with ARP.

See Also arp(1M), ifconfig(1M), privileges(5), if_tcp(7P), inet(7P), netinfo(9F)

Plummer, Dave, *An Ethernet Address Resolution Protocol* or *Converting Network Protocol Addresses to 48 .bit Ethernet Addresses for Transmission on Ethernet Hardware*, RFC 826, STD 0037, November 1982.

Malkin, Gary, *ARP Extension - UNARP, RFC 1868*, November, 1995

Diagnostics Several messages can be written to the system logs (by the IP module) when errors occur. In the following examples, the hardware address strings include colon (:) separated ASCII representations of the link layer addresses, whose lengths depend on the underlying media (for example, 6 bytes for Ethernet).

Node %x:%x ... %x:%x is using our IP address %d.%d.%d.%d on %s.
Duplicate IP address warning. ARP has discovered another host on a local network that responds to mapping requests for the Internet address of this system, and has defended the system against this node by re-announcing the ARP entry.

%s has duplicate address %d.%d.%d.%d (in use by %x:%x ... %x:%x); disabled.
Duplicate IP address detected while performing initial probing. The newly-configured interface has been shut down.

%s has duplicate address %d.%d.%d.%d (claimed by %x:%x ... %x:%x); disabled.
Duplicate IP address detected on a running IP interface. The conflict cannot be resolved, and the interface has been disabled to protect the network.

Recovered address %d.%d.%d.%d on %s.
An interface with a previously-conflicting IP address has been recovered automatically and reenabled. The conflict has been resolved.

Proxy ARP problem? Node '%x:%x ... %x:%x' is using %d.%d.%d.%d on %s
This message appears if arp(1M) has been used to create a published permanent (ATF_AUTHORITY) entry, and some other host on the local network responds to mapping requests for the published ARP entry.

Name asy – asynchronous serial port driver

Synopsis

```
#include <fcntl.h>

#include <sys/termios.h>

open("/dev/term/n", mode);

open("/dev/tty/n", mode);

open("/dev/cua/n", mode);
```

Description The `asy` module is a loadable STREAMS driver that provides basic support for Intel-8250, National Semiconductor-16450, 16550, and some 16650 and 16750 and equivalent UARTs connected via the ISA-bus, in addition to basic asynchronous communication support. The `asy` module supports those `termio(7I)` device control functions specified by flags in the `c_cflag` word of the `termios` structure, and by the `IGNBRK, IGNPAR, PARMRK`, `INPCK`, `IXON`, `IXANY`, or `IXOFF` flags in the c_iflag word of the termios structure. All other `termio(7I)` functions must be performed by STREAMS modules pushed atop the driver. When a device is opened, the `ldterm(7M)` and `ttcompat(7M)` STREAMS modules are automatically pushed on top of the stream, providing the standard `termio(7I)` interface.

The character-special devices `/dev/term/a`, `/dev/term/b`, `/dev/term/c` and `/dev/term/d` are used to access the four standard serial ports (COM1, COM2, COM3 and COM4 at I/O addresses 3f8, 2f8, 3e8 and 2e8 respectively). Serial ports on non-standard ISA-bus I/O addresses are accessed via the character-special devices `/dev/term/0`, `/dev/term/1`, etc. Device names are typically used to provide a logical access point for a dial-in line that is used with a modem.

To allow a single tty line to be connected to a modem and used for incoming and outgoing calls, a special feature is available that is controlled by the minor device number. By accessing character-special devices with names of the form `/dev/cua/`*n*, it is possible to open a port without the `Carrier Detect` signal being asserted, either through hardware or an equivalent software mechanism. These devices are commonly known as dial-out lines.

Note – This module is affected by the setting of certain eeprom variables, ttya-ignore-cd and ttya-rts-dtr-off (and similarly for ttyb-, ttyc-, and ttyd- parameters). For information on these parameters, see the `eeprom(1M)` man page.

Note – For serial ports on the standard COM1 to COM4 I/O addresses above, the default setting for ttya-ignore-cd and ttya-rts-dtr-off is true. If any of these ports are connected to a modem, these settings should be changed to false. For serial ports on non-standard I/O addresses, the default setting for ttya-ignore-cd and ttya-rts-dtr-off is false.

Application Programming Interface Once a `/dev/cua/`*n* line is opened, the corresponding tty line cannot be opened until the `/dev/cua/`*n* line is closed. A blocking open will wait until the `/dev/cua/`*n* line is closed (which will drop `Data Terminal Ready`, after which `Carrier Detect` will usually drop as well) and carrier is detected again. A non-blocking open will return an error. If the `/dev/ttyd`*n* line has been opened successfully (usually only when carrier is recognized on the

modem), the corresponding /dev/cua/*n* line cannot be opened. This allows a modem to be attached to /dev/term/[*n*] (renamed from /dev/tty[*n*]) and used for dial-in (by enabling the line for login in /etc/inittab) or dial-out (by tip(1) or uucp(1C)) as /dev/cua/*n* when no one is logged in on the line.

ioctls The standard set of termio ioctl() calls are supported by asy.

Breaks can be generated by the TCSBRK, TIOCSBRK, and TIOCCBRK ioctl() calls.

The input and output line speeds may be set to any speed that is supported by termio. The speeds cannot be set independently; for example, when the output speed is set, the input speed is automatically set to the same speed.

When the asy module is used to service the serial console port, it supports a BREAK condition that allows the system to enter the debugger or the monitor. The BREAK condition is generated by hardware and it is usually enabled by default.

A BREAK condition originating from erroneous electrical signals cannot be distinguished from one deliberately sent by remote DCE. The Alternate Break sequence can be used as a remedy against this. Due to a risk of incorrect sequence interpretation, SLIP and certain other binary protocols should not be run over the serial console port when Alternate Break sequence is in effect. Although PPP is a binary protocol, it is able to avoid these sequences using the ACCM feature in *RFC 1662*. For Solaris PPP 4.0, you do this by adding the following line to the /etc/ppp/options file (or other configuration files used for the connection; see pppd(1M) for details):

```
asyncmap 0x00002000
```

By default, the Alternate Break sequence is a three character sequence: carriage return, tilde and control-B (CR ~ CTRL-B), but may be changed by the driver. For more information on breaking (entering the debugger or monitor), see kbd(1) and kb(7M).

Errors An open() will fail under the following conditions:

ENXIO The unit being opened does not exist.

EBUSY The dial-out device is being opened while the dial-in device is already open, or the dial-in device is being opened with a no-delay open and the dial-out device is already open.

EBUSY The unit has been marked as exclusive-use by another process with a TIOCEXCL ioctl() call.

EINTR The open was interrupted by the delivery of a signal.

Files

/dev/term/[*a-d*]	
/dev/term/[*012...*]	dial-in tty lines
/dev/cua/[*a-d*]	
/dev/cua/[*012...*]	dial-out tty lines

`/kernel/drv/amd64/asy`	64-bit kernel module for 64–bit x86 platform
`/kernel/drv/asy.conf`	`asy` configuration file

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86

See Also `tip(1)`, `kbd(1)`, `uucp(1C)`, `eeprom(1M)`, `pppd(1M)`, `ioctl(2)`, `open(2)`, `termios(3C)`, `attributes(5)`, `ldterm(7M)`, `ttcompat(7M)`, `kb(7M)`, `termio(7I)`

Diagnostics

`asy`*n* `: silo overflow.`	The hardware overrun occurred before the input character could be serviced.
`asy`*n* `: ring buffer overflow.`	The driver's character input ring buffer overflowed before it could be serviced.

Name ata – AT attachment disk driver

Synopsis `ide@unit-`*address*

Description The `ata` driver supports disk and ATAPI CD/DVD devices conforming to the AT Attachment specification including IDE interfaces. Support is provided for both parallel ATA (PATA) and serial ATA (SATA) interfaces.

Refer to the *Solaris x86 Hardware Compatibility List* for a list of supported controllers.

Preconfigure A PCI IDE controller can operate in compatibility mode or in PCI-native mode. If more than one controller is present in the system, only one can operate in compatibility mode.

If two PATA drives share the same controller, you must set one to master and the other to slave. If both a PATA disk drive and a PATA CD-ROM drive utilize the same controller, you can designate the disk drive as the master with the CD-ROM drive as the slave, although this is not mandatory.

Supported Settings

Supported settings for the primary controller when in compatibility mode are:

- IRQ Level: 14
- I/O Address: 0x1F0

Supported settings for the secondary controller when in compatibility mode are:

- IRQ Level: 15
- I/O Address: 0x170

Note – When in PCI-native mode, the IRQ and I/O address resources are configured by the system BIOS.

Known Problems and Limitations

- This driver does not support any RAID features present on a PATA/SATA controller. As a result, you should configure BIOS to select IDE mode rather than RAID mode. Some systems may require updating BIOS to allow switching modes.
- On some systems, the SATA controller must have option ROM enabled or BIOS will not consider SATA drives as bootable devices.
- Panasonic LK-MC579B and the Mitsumi FX34005 IDE CD-ROM drives are not supported and cannot be used to install the Solaris operating environment.
- CMD-604 is unable to handle simultaneous I/O on both IDE interfaces. This defect causes the Solaris software to hang if both interfaces are used. Use only the primary IDE interface at address 0x1F0.
- NEC CDR-260/CDR-260R/CDR-273 and Sony CDU-55E ATAPI CD-ROM drives might fail during installation.
- Sony CDU-701 CD-ROM drives must be upgraded to use firmware version 1.0r or later to support booting from the CD.

A Compact Flash(CF) card can work as an ATA disk through a CF-to-ATA adapter. If both card and adapter implement Compact Flash Version 2.0, DMA is supported. If either of them does not, you should set ata-disk-dma-enabled to '0.'

Configuration The ata driver properties are usually set in `ata.conf`. However, it may be convenient, or in some cases necessary, for you to set some of the DMA related properties as a system global boot environment property. You set or modify properties in the boot environment immediately prior to booting the Solaris kernel using the GRUB boot loader kernel boot command line. You can also set boot environment properties using the `eeprom(1M)` command or by editing the `bootenv.rc` configuration file. If a property is set in both the driver's `ata.conf` file and the boot environment, the `ata.conf` property takes precedence.

Property modifications other than with the GRUB kernel boot command line are not effective until you reboot the system. Property modifications via the GRUB kernel boot command line do not persist across future boots.

Direct Memory Access is enabled for disks and atapi CD/DVD by default. If you want to disable DMA when booting from a CD/DVD, you must first set atapi-cd-dma-enabled to 0 using the GRUB kernel boot command line.

`ata-dma-enabled` This property is examined before the DMA properties discussed below. If it is set to '0,' DMA is disabled for all ATA/ATAPI devices, and no further property checks are made. If this property is absent or is set to '1,' DMA status is determined by further examining one of the other properties listed below.

`ata-disk-dma-enabled` This property is examined only for ATA disk devices, and only if ata-dma-enabled is not set to '0.'

If ata-disk-dma-enabled set to '0,' DMA is disabled for all ATA disks in the system. If this property is absent or set to '1,' DMA is enabled for all ATA disks and no further property checks are made. If needed, this property should be created by the administrator using the GRUB kernel boot command line or the `eeprom(1M)` command.

`atapi-cd-dma-enabled` This property is examined only for ATAPI CD/DVD devices, and only if ata-dma-enabled is not set to '0.'

If atapi-cd-dma-enabled is absent or set to '0,' DMA is disabled for all ATAPI CD/DVD's. If set to '1,' DMA is enabled and no further property checks are made.

The Solaris installation program creates this property in the boot environment with a value of '1.' It can be changed with

the GRUB kernel boot command line or eeprom(1M) as shown in the Example section of this manpage.

`atapi-other-dma-enabled`

This property is examined only for non-CD/DVD ATAPI devices such as ATAPI tape drives, and only if ata-dma-enabled is not set to '0.'

If atapi-other-dma-enabled is set to '0,' DMA is disabled for all non-CD/DVD ATAPI devices. If this property is absent or set to '1,' DMA is enabled and no further property checks are made.

If needed, this property should be created by the administrator using the GRUB kernel boot command line or the eeprom(1M) command.

`drive0_block_factor`
`drive1_block_factor`

ATA controllers support some amount of buffering (blocking). The purpose is to interrupt the host when an entire buffer full of data has been read or written instead of using an interrupt for each sector. This reduces interrupt overhead and significantly increases throughput. The driver interrogates the controller to find the buffer size. Some controllers hang when buffering is used, so the values in the configuration file are used by the driver to reduce the effect of buffering (blocking). The values presented may be chosen from `0x1`, `0x2`, `0x4`, `0x8` and `0x10`.

The values as shipped are set to `0x1`, and they can be tuned to increase performance.

If your controller hangs when attempting to use higher block factors, you may be unable to reboot the system. For x86 based systems, it is recommended that tuning be performed using a duplicate of the `/platform/i86pc/kernel` directory subtree. This ensures that a bootable kernel subtree exists in the event of a failed test.

`ata-revert-to-defaults`
`revert–<diskmodel>`

When rebooting or shutting down, the driver can set a feature which allows the drive to return to the power-on settings when the drive receives a software reset (SRST) sequence. If this property is present and set to 1, the driver will set the feature to revert to defaults during reset. Setting this property to 1 may prevent some systems from soft-rebooting and

would require cycling the power to boot the system. If this property is not present the system will not set the feature to revert to defaults during reset.

To determine the string to substitute for <diskmodel>, boot your system (you may have to press the reset button or power-cycle) and then view `/var/adm/messages`. Look for the string "IDE device at targ" or "ATAPI device at targ." The next line will contain the word "model" followed by the model number and a comma. Ignore all characters except letters, digits, ".", "_", and "-". Change uppercase letters to lower case. If the string revert-<diskmodel> is longer than 31 characters, use only the first 31 characters.

Examples

EXAMPLE 1 Sample ata Configuration File

```
# for higher performance - set block factor to 16
    drive0_block_factor=0x1 drive1_block_factor=0x1
    max_transfer=0x100
    flow_control="dmult" queue="qsort" disk="dadk" ;
```

EXAMPLE 2 Revert to defaults property

```
revert-st320420a=1;
```

Output of `/var/adm/messages`:

```
Aug 17 06:49:43 caesar ata:[ID 640982 kern.info] IDE device at targ 0,
                           lun 0 lastlun 0x0
Aug 17 06:49:43 caesar ata:[ID 521533 kern.info] model ST320420A, stat
```

EXAMPLE 3 Change DMA property using GRUB

To change a DMA property using the GRUB kernel boot command line:

1. Reset the system.
2. Press “e” to interrupt the timeout.
3. Select the kernel line.
4. Press “e.”
5. If there is no existing -B option:

   Add: -B atapi-cd-dma-enabled=1

   else...

   Add: atapi-cd-dma-enabled=1 to the end of the current -B option. For example:-B foo=bar,atapi-cd-dma-enabled=1.

EXAMPLE 3 Change DMA property using GRUB *(Continued)*

6. Press Enter to commit the edited line to memory. (Does not write to the disk and is non-persistent).
7. Press 'b' to boot the modified entry.

EXAMPLE 4 Change DMA Property with eeprom(1M)

To enable DMA for optical devices while the Solaris kernel is running with the `eeprom(1M)` system command:

```
eeprom 'atapi-cd-dma-enabled=1'
```

Files

`/platform/i86pc/kernel/drv/ata`	Device driver.
`/platform/i86pc/kernel/drv/ata.conf`	Configuration file.
`/boot/solaris/bootenv.rc`	Boot environment variables file for Solaris x86. `eeprom(1M)` can be used to modify properties in this file.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86

See Also `eeprom(1M)`, `attributes(5)`, `grub(5)`

INCITS T13 ATA/ATAPI-7 specifications

Name atge – Device driver for Atheros/Attansic Ethernet chipsets

Description The `atge` ethernet driver is GLD based supporting the Atheros/Attansic L1E Gigabit Ethernet 10/100/1000 Base (AR8121/AR8113) chipsets:

```
pciex1969,1026 Atheros/Attansic GigabitE 10/100/1000 Base (AR8121/AR8113)
```

The `atge` driver supports IEEE 802.3 auto-negotiation, flow control and VLAN tagging.

Configuration The default configuration is auto-negotiation with bi-directional flow control. The advertised capabilities for auto-negotiation are based on the capabilities of the `PHY`.

You can set the capabilities advertised by the `atge` controlled device using `dladm`(1M). The driver supports only those parameters which begin with en (enabled) in the parameters listed by the command `dladm`(1M). Each of these boolean parameters determines if the device advertises that mode of operation when the hardware supports it.

Files

`/dev/atge`	Special character device
`/kernel/drv/atge`	32-bit device drive (x86)
`/kernel/drv/amd64/atge`	64-bit device driver (x86)

Attributes See `attributes`(5) for a description of the following attribute:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86

See Also `dladm`(1M), `ndd`(1M), `netstat`(1M), `driver.conf`(4), `attributes`(5), `ieee802.3`(5), `dlpi`(7P), `streamio`(7I)

Writing Device Drivers

Network Interface Guide

STREAMS Programmer's Guide

IEEE 802.3ae Specification, 2002

Name ath – Atheros AR52xx 802.11b/g wireless NIC driver

Description The `ath` *802.11b/g* wireless NIC driver is a multi-threaded, loadable, clonable, GLDv3-based STREAMS driver for the Atheros AR52xx (AR5210/5211/5212) chipset-based wireless NIC.

The `ath` driver functions include controller initialization, wireless *802.11b/g* infrastructure network connection, WEP, frame transmit and receive, and promiscuous and multi-cast support.

Driver Configuration The `ath` driver performs auto-negotiation to determine the data rates and mode. Supported *802.11b* data rates (Mbits/sec.) are 1, 2, 5.5 and 11. Supported *802.11g* data rates (Mbits/sec.) are 1, 2, 5.5, 11, 6, 9, 12, 18, 24, 36, 48 and 54.

The `ath` driver supports only BSS networks (also known as "ap" or "infrastructure" networks) and the "open" (or "opensystem") authentication type. Only WEP encryption is currently supported. Configuration and administration can be performed through the `dladm(1M)` or `wificonfig(1M)` utilities.

Files

`/dev/ath*`	Special character device.
`/kernel/drv/ath`	32-bit ELF kernel module (x86).
`/kernel/drv/amd64/ath`	64-bit ELF kernel module (x86).

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86

See Also `dladm(1M)`, `wificonfig(1M)`, `gld(7D)`

ANSI/IEEE Std 802.11– Standard for Wireless LAN Technology, 1999.

IEEE Std 802.11a– Standard for Wireless LAN Technology-Rev. A, 2003

IEEE Std 802.11b - Standard for Wireless LAN Technology-Rev.B, 2003

IEEE Std 802.11g— Standard for Wireless LAN Technology - Rev. G, 2003

Name atu – Atmel AT76C50x USB IEEE 802.11b Wireless Device Driver

Description The `atu` `802.11b` wireless driver is a multi-threaded, loadable, clonable, GLDv3-based STREAMS driver supporting the Atmel AT76C50x chipset-based wireless devices.

Configuration The `atu` driver performs auto-negotiation to determine the data rate and mode. Supported 802.11b data rates are 1, 2, 5.5, and 11 Mbits/sec.

The `atu` driver supports only BSS networks (also known as `ap` or `infrastructure` networks).

`open` (or `open-system`) and `shared` `key` authentication modes are supported. Encryption types WEP40 and WEP104 are supported.

Files

`/dev/atu*`	Special character device
`/kernel/drv/atu`	32-bit ELF kernel module (x86)
`/kernel/drv/amd64/atu`	64-bit ELF kernel module (x86)

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWatu
Interface Stability	Committed

See Also `dladm(1M)`, `wificonfig(1M)`, `attributes(5)`, `dlpi(7P)`, `gld(7D)`

802.11 - Wireless LAN Media Access Control and Physical Layer Specification - IEEE, 2001

Name audio1575 – Uli M1575 Super South Bridge audio digital controller interface

Description The `audio1575` device uses the Uli M1575 AC97–compatible audio digital controller and an AC-97 Codec to implement the audio device interface. This interface allows analog only inputs and outputs.

Files

`/kernel/drv/sparcv9/audio1575`	64–bit driver module
`/kernel/drv/audio1575.conf`	Driver configuration file

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86
Availability	SUNWaudd
Stability level	Uncommitted

See Also `ioctl(2)`, `attributes(5)`, `audio(7I)`, `mixer(7I)`, `streamio(7I)`

Uli M1575 Super South Bridge Data Sheet Data Sheet— Uli USA Inc.

AD1981B AC '97 SoundMAX(R) Codec Data Sheet— Analog Devices Inc.

Diagnostics In addition to being logged, the following messages might appear on the system console:

`play-interrupts too low`
`record-interrupts too low` — The interrupt rate specified in `audio1575.conf` is set too low. It has been reset to the rate specified in the message. Update `audio1575.conf` to a higher interrupt rate.

`play-interrupts too high`
`record-interrupts too high` — The interrupt rate specified in `audio1575.conf` is set too high. It has been reset to the rate specified in the message. Update `audio1575.conf` to a lower interrupt rate.

Name audio – common audio framework

Description The `audio` driver provides common support routines for audio devices in Solaris.

The audio framework supports multiple *personalities*, allowing for devices to be accessed with different programming interfaces.

The audio framework also provides a number of facilities, such as mixing of audio streams, and data format and sample rate conversion.

Overview The audio framework provides a software mixing engine (audio mixer) for all audio devices, allowing more than one process to play or record audio at the same time.

Multi-Stream Codecs

The audio mixer supports multi-stream Codecs. These devices have DSP engines that provide sample rate conversion, hardware mixing, and other features. The use of such hardware features is opaque to applications.

Backward Compatibility

It is not possible to disable the mixing function. Applications must not assume that they have exclusive access to the audio device.

Audio Formats Digital audio data represents a quantized approximation of an analog audio signal waveform. In the simplest case, these quantized numbers represent the amplitude of the input waveform at particular sampling intervals. To achieve the best approximation of an input signal, the highest possible sampling frequency and precision should be used. However, increased accuracy comes at a cost of increased data storage requirements. For instance, one minute of monaural audio recorded in u-Law format (pronounced *mew-law*) at 8 KHz requires nearly 0.5 megabytes of storage, while the standard Compact Disc audio format (stereo 16-bit linear PCM data sampled at 44.1 KHz) requires approximately 10 megabytes per minute.

An audio data format is characterized in the audio driver by four parameters: sample Rate, encoding, precision, and channels. Refer to the device-specific manual pages for a list of the audio formats that each device supports. In addition to the formats that the audio device supports directly, other formats provide higher data compression. Applications can convert audio data to and from these formats when playing or recording.

Sample Rate

Sample rate is a number that represents the sampling frequency (in samples per second) of the audio data.

The audio mixer always configures the hardware for the highest possible sample rate for both play and record. This ensures that none of the audio streams require compute-intensive low pass filtering. The result is that high sample rate audio streams are not degraded by filtering.

Sample rate conversion can be a compute-intensive operation, dependingon the number of channels and a device's sample rate. For example, an 8KHz signal can be easily converted to 48KHz, requiring a low cost up sampling by 6. However, converting from 44.1KHz to 48KHz is computer intensive because it must be up sampled by 160 and then down sampled by 147. This is only done using integer multipliers.

Applications can greatly reduce the impact of sample rate conversion by carefully picking the sample rate. Applications should always use the highest sample rate the device supports. An application can also do its own sample rate conversion (to take advantage of floating point and accelerated instructions) or use small integers for up and down sampling.

All modern audio devices run at 48 kHz or a multiple thereof, hence just using 48 kHz can be a reasonable compromise if the application is not prepared to select higher sample rates.

Encodings

An encoding parameter specifies the audiodata representation. u-Law encoding corresponds to CCITT G.711, and is the standard for voice data used by telephone companies in the United States, Canada, and Japan. A-Law encoding is also part of CCITT G.711 and is the standard encoding for telephony elsewhere in the world. A-Law and u-Law audio data are sampled at a rate of 8000 samples per second with 12-bit precision, with the data compressed to 8-bit samples. The resulting audio data quality is equivalent to that of stan dard analog telephone service.

Linear Pulse Code Modulation (PCM) is an uncompressed, signed audio format in which sample values are directly proportional to audio signal voltages. Each sample is a 2's complement number that represents a positive or negative amplitude.

Precision

Precision indicates the number of bits used to store each audio sample. For instance, u-Law and A-Law data are stored with 8-bit precision. PCM data can be stored at various precisions, though 16-bit is the most common.

Channels

Multiple channels of audio can be interleaved at sample boundaries. A sample frame consists of a single sample from each active channel. For example, a sample frame of stereo 16-bit PCM data consists of 2 16-bit samples, corresponding to the left and right channel data. The audio mixer sets the hardware to the maximum number of channels supported. If a mono signal is played or recorded, it is mixed on the first two (usually the left and right) channel only. Silence is mixed on the remaining channels.

Supported Formats

The audio mixer supports the following audio formats:

```
Encoding            Precision  Channels
Signed Linear PCM   32-bit     Mono or Stereo
Signed Linear PCM   16-bit     Mono or Stereo
Signed Linear PCM   8-bit      Mono or Stereo
u-Law               8-bit      Mono or Stereo
A-Law               8-bit      Mono or Stereo
```

The audio mixer converts all audio streams to 24-bit Linear PCM before mixing. After mixing, conversion is made to the best possible Codec format. The conversion process is not compute intensive and audio applications can choose the encoding format that best meets their needs.

The mixer discards the low order 8 bits of 32-bit Signed Linear PCM in order to perform mixing. (This is done to allow for possible overflows to fit into 32-bits when mixing multiple streams together.) Hence, the maximum effective precision is 24-bits.

Files

`/kernel/drv/audio`	32-bit kernel driver module
`/kernel/drv/amd64/audio`	64-bit x86 kernel driver module
`/kernel/drv/sparcv9/audio`	64-bit SPARC kernel driver module
`/kernel/drv/audio.conf`	`audio` configuration file

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86
Availability	SUNWaudd
Interface Stability	Uncommitted

See Also `ioctl(2)`, `attributes(5)`, `audio(7I)`, `dsp(7I)`

Name audio – generic audio device interface

Synopsis

```
#include <sys/audio.h>
```

Overview An audio device is used to play and/or record a stream of audio data. Since a specific audio device may not support all functionality described below, refer to the device-specific manual pages for a complete description of each hardware device. An application can use the `AUDIO_GETDEV` `ioctl(2)` to determine the current audio hardware associated with `/dev/audio`.

The audio framework provides a software mixing engine (audio mixer) for all audio devices, allowing more than one process to play or record audio at the same time.

Backward Compatibility It is no longer possible to disable the mixing function. Applications must not assume that they have exclusive access to the audio device.

Multi-Stream Codecs The audio mixer supports multi-stream Codecs. These devices have DSP engines that provide sample rate conversion, hardware mixing, and other features. The use of such hardware features is opaque to applications.

Audio Formats Digital audio data represents a quantized approximation of an analog audio signal waveform. In the simplest case, these quantized numbers represent the amplitude of the input waveform at particular sampling intervals. To achieve the best approximation of an input signal, the highest possible sampling frequency and precision should be used. However, increased accuracy comes at a cost of increased data storage requirements. For instance, one minute of monaural audio recorded in µ-Law format (pronounced *mew-law*) at 8 KHz requires nearly 0.5 megabytes of storage, while the standard Compact Disc audio format (stereo 16-bit linear PCM data sampled at 44.1 KHz) requires approximately 10 megabytes per minute.

Audio data may be represented in several different formats. An audio device's current audio data format can be determined by using the `AUDIO_GETINFO` `ioctl(2)` described below.

An audio data format is characterized in the audio driver by four parameters: Sample Rate, Encoding, Precision, and Channels. Refer to the device-specific manual pages for a list of the audio formats that each device supports. In addition to the formats that the audio device supports directly, other formats provide higher data compression. Applications may convert audio data to and from these formats when playing or recording.

Sample Rate Sample rate is a number that represents the sampling frequency (in samples per second) of the audio data.

The audio mixer always configures the hardware for the highest possible sample rate for both play and record. This ensures that none of the audio streams require compute-intensive low pass filtering. The result is that high sample rate audio streams are not degraded by filter ing.

Sample rate conversion can be a compute-intensive operation, depending on the number of channels and a device's sample rate. For example, an 8KHz signal can be easily converted to

48KHz, requiring a low cost up sampling by 6. However, converting from 44.1KHz to 48KHz is compute intensive because it must be up sampled by 160 and then down sampled by 147. This is only done using integer multipliers.

Applications can greatly reduce the impact of sample rate conversion by carefully picking the sample rate. Applications should always use the highest sample rate the device supports. An application can also do its own sample rate conversion (to take advantage of floating point and accelerated instruction or use small integers for up and down sampling.

All modern audio devices run at 48 kHz or a multiple thereof, hence just using 48 kHz may be a reasonable compromise if the application is not prepared to select higher sample rates.

Encodings

An encoding parameter specifies the audio data representation. µ-Law encoding corresponds to *CCITT G.711*, and is the standard for voice data used by telephone companies in the United States, Canada, and Japan. A-Law encoding is also part of *CCITT G.711* and is the standard encoding for telephony elsewhere in the world. A-Law and µ-Law audio data are sampled at a rate of 8000 samples per second with 12-bit precision, with the data compressed to 8-bit samples. The resulting audio data quality is equivalent to that of standard analog telephone service.

Linear Pulse Code Modulation (PCM) is an uncompressed, signed audio format in which sample values are directly proportional to audio signal voltages. Each sample is a 2's complement number that represents a positive or negative amplitude.

Precision

Precision indicates the number of bits used to store each audio sample. For instance, u-Law and A-Law data are stored with 8-bit precision. PCM data may be stored at various precisions, though 16-bit is the most common.

Channels

Multiple channels of audio may be interleaved at sample boundaries. A sample frame consists of a single sample from each active channel. For example, a sample frame of stereo 16-bit PCM data consists of two 16-bit samples, corresponding to the left and right channel data.

The audio mixer sets the hardware to the maximum number of channels supported. If a mono signal is played or recorded, it is mixed on the first two (usually the left and right) channels only. Silence is mixed on the remaining channels

Supported Formats

The audio mixer supports the following audio formats:

```
Encoding           Precision   Channels
Signed Linear PCM  32-bit      Mono or Stereo
Signed Linear PCM  16-bit      Mono or Stereo
Signed Linear PCM   8-bit      Mono or Stereo
u-Law               8-bit      Mono or Stereo
A-Law               8-bit      Mono or Stereo
```

The audio mixer converts all audio streams to 24-bit Linear PCM before mixing. After mixing, conversion is made to the best possible Codec format. The conversion process is not compute intensive and audio applications can choose the encoding format that best meets their needs.

Note that the mixer discards the low order 8 bits of 32-bit Signed Linear PCM in order to perform mixing. (This is done to allow for possible overflows to fit into 32-bits when mixing multiple streams together.) Hence, the maximum effective precision is 24-bits.

Description

The device `/dev/audio` is a device driver that dispatches audio requests to the appropriate underlying audio hardware. The audio driver is implemented as a STREAMS driver. In order to record audio input, applications `open(2)` the `/dev/audio` device and read data from it using the `read(2)` system call. Similarly, sound data is queued to the audio output port by using the `write(2)` system call. Device configuration is performed using the `ioctl(2)` interface.

Because some systems may contain more than one audio device, application writers are encouraged to query the `AUDIODEV` environment variable. If this variable is present in the environment, its value should identify the path name of the default audio device.

Opening the Audio Device

The audio device is not treated as an exclusive resource. Each process may open the audio device once.

Each `open()` completes as long as there are channels available to be allocated. If no channels are available to be allocated:

- if either the `O_NDELAY` or `O_NONBLOCK` flags are set in the `open()` *oflag* argument, then –1 is immediately returned, with *errno* set to `EBUSY`.
- if neither the `O_NDELAY` nor the `O_NONBLOCK` flag are set, then `open()` hangs until the device is available or a signal is delivered to the process, in which case a –1 is returned with *errno* set to `EINTR`.

Upon the initial open() of the audio channel, the audio mixer sets the data format of the audio channel to the default state of 8-bit, 8Khz, mono u-Law data. If the audio device does not support this configuration, it informs the audio mixer of the initial configuration. Audio applications should explicitly set the encoding characteristics to match the audio data requirements, and not depend on the default configuration.

Recording Audio Data

The `read()` system call copies data from the system's buffers to the application. Ordinarily, `read()` blocks until the user buffer is filled. The `I_NREAD` `ioctl` (see `streamio(7I)`) may be used to determine the amount of data that may be read without blocking. The device may alternatively be set to a non-blocking mode, in which case `read()` completes immediately, but may return fewer bytes than requested. Refer to the `read(2)` manual page for a complete description of this behavior.

When the audio device is opened with read access, the device driver immediately starts buffering audio input data. Since this consumes system resources, processes that do not record audio data should open the device write-only (`O_WRONLY`).

The transfer of input data to STREAMS buffers may be paused (or resumed) by using the `AUDIO_SETINFO` `ioctl` to set (or clear) the *record.pause* flag in the audio information structure (see below). All unread input data in the STREAMS queue may be discarded by using the `I_FLUSH` STREAMS `ioctl`. See `streamio(7I)`. When changing record parameters, the input

stream should be paused and flushed before the change, and resumed afterward. Otherwise, subsequent reads may return samples in the old format followed by samples in the new format. This is particularly important when new parameters result in a changed sample size.

Input data can accumulate in STREAMS buffers very quickly. At a minimum, it will accumulate at 8000 bytes per second for 8-bit, 8 KHz, mono, u-Law data. If the device is configured for 16-bit linear or higher sample rates, it will accumulate even faster. If the application that consumes the data cannot keep up with this data rate, the STREAMS queue may become full. When this occurs, the *record.error* flag is set in the audio information structure and input sampling ceases until there is room in the input queue for additional data. In such cases, the input data stream contains a discontinuity. For this reason, audio recording applications should open the audio device when they are prepared to begin reading data, rather than at the start of extensive initialization.

Playing Audio Data

The `write()` system call copies data from an application's buffer to the STREAMS output queue. Ordinarily, `write()` blocks until the entire user buffer is transferred. The device may alternatively be set to a non-blocking mode, in which case `write()` completes immediately, but may have transferred fewer bytes than requested. See `write(2)`.

Although `write()` returns when the data is successfully queued, the actual completion of audio output may take considerably longer. The `AUDIO_DRAIN` `ioctl` may be issued to allow an application to block until all of the queued output data has been played. Alternatively, a process may request asynchronous notification of output completion by writing a zero-length buffer (end-of-file record) to the output stream. When such a buffer has been processed, the *play.eof* flag in the audio information structure is incremented.

The final `close(2)` of the file descriptor hangs until all of the audio output has drained. If a signal interrupts the `close()`, or if the process exits without closing the device, any remaining data queued for audio output is flushed and the device is closed immediately.

The consumption of output data may be paused (or resumed) by using the `AUDIO_SETINFO` `ioctl` to set (or clear) the *play.pause* flag in the audio information structure. Queued output data may be discarded by using the `I_FLUSH` STREAMS `ioctl`. (See `streamio(7I)`).

Output data is played from the STREAMS buffers at a default rate of at least 8000 bytes per second for µ-Law, A-Law or 8–bit PCM data (faster for 16-bit linear data or higher sampling rates). If the output queue becomes empty, the *play.error* flag is set in the audio information structure and output is stopped until additional data is written. If an application attempts to write a number of bytes that is not a multiple of the current sample frame size, an error is generated and the bad data is thrown away. Additional writes are allowed.

Asynchronous I/O

The `I_SETSIG` STREAMS `ioctl` enables asynchronous notification, through the `SIGPOLL` signal, of input and output ready condition changes. The `O_NONBLOCK` flag may be set using the `F_SETFL` `fcntl(2)` to enable non-blocking `read()` and `write()` requests. This is normally sufficient for applications to maintain an audio stream in the background.

Audio Control Pseudo-Device

It is sometimes convenient to have an application, such as a volume control panel, modify certain characteristics of the audio device while it is being used by an unrelated process.

The `/dev/audioctl` pseudo-device is provided for this purpose. Any number of processes may open `/dev/audioctl` simultaneously. However, `read()` and `write()` system calls are ignored by `/dev/audioctl`. The `AUDIO_GETINFO` and `AUDIO_SETINFO` `ioctl` commands may be issued to `/dev/audioctl` to determine the status or alter the behavior of `/dev/audio`. Note: In general, the audio control device name is constructed by appending the letters "`ctl`" to the path name of the audio device.

Audio Status Change Notification

Applications that open the audio control pseudo-device may request asynchronous notification of changes in the state of the audio device by setting the `S_MSG` flag in an `I_SETSIG` STREAMS `ioctl`. Such processes receive a `SIGPOLL` signal when any of the following events occur:

- An `AUDIO_SETINFO` `ioctl` has altered the device state.
- An input overflow or output underflow has occurred.
- An end-of-file record (zero-length buffer) has been processed on output.
- An `open()` or `close()` of `/dev/audio` has altered the device state.
- An external event (such as speakerbox's volume control) has altered the device state.

ioctls

Audio Information Structure

The state of the audio device may be polled or modified using the `AUDIO_GETINFO` and `AUDIO_SETINFO` `ioctl` commands. These commands operate on the `audio_info` structure as defined, in `<sys/audio.h>`, as follows:

```
/*
* This structure contains state information for audio device
* IO streams
*/

struct audio_prinfo {
  /*
  * The following values describe the
  * audio data encoding
  */
 uint_t sample_rate; /* samples per second */
 uint_t channels;  /* number of interleaved channels */
 uint_t precision; /* number of bits per sample */
 uint_t encoding;  /* data encoding method */


  /*
  * The following values control audio device
  * configuration
  */
```

```
 uint_t gain; /* volume level */
 uint_t port; /* selected I/O port */
 uint_t buffer_size; /* I/O buffer size */

  /*
  * The following values describe the current device
  * state
  */
 uint_t samples; /* number of samples converted */
 uint_t eof;  /* End Of File counter (play only) */
 uchar_t pause;  /* non-zero if paused, zero to resume */
 uchar_t error; /* non-zero if overflow/underflow */
 uchar_t waiting; /* non-zero if a process wants access */
 uchar_t balance; /* stereo channel balance */
   /*
   * The following values are read-only device state
   * information
   */
 uchar_t open;/* non-zero if open access granted */
 uchar_t active; /* non-zero if I/O active */
 uint_t avail_ports; /* available I/O ports */
 uint_t mod_ports; /* modifiable I/O ports */
};
typedef struct audio_prinfo audio_prinfo_t;

  /*
  * This structure is used in AUDIO_GETINFO and AUDIO_SETINFO ioctl
  * commands
  */
struct audio_info {
 audio_prinfo_t record;/* input status info */
 audio_prinfo_t play;/* output status info */
 uint_t monitor_gain; /* input to output mix */
 uchar_toutput_muted; /* non-zero if output muted */
 uint_t hw_features; /* supported H/W features */
 uint_t sw_features;/* supported S/W features */
 uint_t sw_features_enabled;
                /* supported S/W features enabled */
};
typedef struct audio_info audio_info_t;

/* Audio encoding types */
#define AUDIO_ENCODING_ULAW (1) /* u-Law encoding */
#define AUDIO_ENCODING_ALAW (2) /* A-Law encoding */
#define AUDIO_ENCODING_LINEAR (3) /* Signed Linear PCM encoding */
/*
* These ranges apply to record, play, and
* monitor gain values
```

```
 */
#define AUDIO_MIN_GAIN (0)/* minimum gain value */
#define AUDIO_MAX_GAIN (255) /* maximum gain value */

/*
* These values apply to the balance field to adjust channel
* gain values
 */
#define AUDIO_LEFT_BALANCE(0) /* left channel only */
#define AUDIO_MID_BALANCE (32) /* equal left/right balance */
#define AUDIO_RIGHT_BALANCE (64) /* right channel only */

/*
* Define some convenient audio port names
* (for port, avail_ports and mod_ports)
 */

/* output ports (several might be enabled at once) */
#define AUDIO_SPEAKER (0x01)/* built-in speaker */
#define AUDIO_HEADPHONE (0x02)/* headphone jack */
#define AUDIO_LINE_OUT (0x04)/* line out */
#define AUDIO_SPDIF_OUT (0x08)/* SPDIF port */
#define AUDIO_AUX1_OUT (0x10)/* aux1 out */
#define AUDIO_AUX2_OUT (0x20)/* aux2 out */

/* input ports (usually only one may be
* enabled at a time)
 */
#define AUDIO_MICROPHONE (0x01) /* microphone */
#define AUDIO_LINE_IN (0x02) /* line in */
#define AUDIO_CD(0x04) /* on-board CD inputs */
#define AUDIO_SPDIF_IN (0x08) /* SPDIF port */
#define AUDIO_AUX1_IN (0x10) /* aux1 in */
#define AUDIO_AUX2_IN (0x20) /* aux2 in */
#define AUDIO_CODEC_LOOPB_IN (0x40) /* Codec inter.loopback */

/* These defines are for hardware features */
#define AUDIO_HWFEATURE_DUPLEX (0x00000001u)
/*simult. play & cap. supported */

#define AUDIO_HWFEATURE_MSCODEC (0x00000002u)
/* multi-stream Codec */

/* These defines are for software features *
#define AUDIO_SWFEATURE_MIXER (0x00000001u)
 /* audio mixer audio pers. mod. */

/*
```

```
* Parameter for the AUDIO_GETDEV ioctl
* to determine current audio devices
*/#define MAX_AUDIO_DEV_LEN(16)
struct audio_device {
 char name[MAX_AUDIO_DEV_LEN];
 char version[MAX_AUDIO_DEV_LEN];
 char config[MAX_AUDIO_DEV_LEN];
};
typedef struct audio_device audio_device_t;
```

The *play.gain* and *record.gain* fields specify the output and input volume levels. A value of `AUDIO_MAX_GAIN` indicates maximum volume. Audio output may also be temporarily muted by setting a non-zero value in the *output_muted* field. Clearing this field restores audio output to the normal state.

The *monitor_gain* field is present for compatibility, and is no longer supported. See `dsp(7I)` for more detail.

Likewise, the *play.port*, *play.ports*, *play.mod_ports*, *record.port*, *record.ports*, and *record.mod_ports* are no longer supported. See `dsp(7I)` for more detail.

The *play.balance* and *record.balance* fields are fixed to `AUDIO_MID_BALANCE`. Changes to volume levels for different channels can be made using the interfaces in `dsp(7I)`.

The *play.pause* and *record.pause* flags may be used to pause and resume the transfer of data between the audio device and the STREAMS buffers. The *play.error* and *record.error* flags indicate that data underflow or overflow has occurred. The *play.active* and *record.active* flags indicate that data transfer is currently active in the corresponding direction.

The *play.open* and *record.open* flags indicate that the device is currently open with the corresponding access permission. The *play.waiting* and *record.waiting* flags provide an indication that a process may be waiting to access the device. These flags are set automatically when a process blocks on `open()`, though they may also be set using the `AUDIO_SETINFO` ioctl command. They are cleared only when a process relinquishes access by closing the device.

The *play.samples* and *record.samples* fields are zeroed at `open()` and are incremented each time a data sample is copied to or from the associated STREAMS queue. Some audio drivers may be limited to counting buffers of samples, instead of single samples for their samples accounting. For this reason, applications should not assume that the samples fields contain a perfectly accurate count. The *play.eof* field increments whenever a zero-length output buffer is synchronously processed. Applications may use this field to detect the completion of particular segments of audio output.

The *record.buffer_size* field controls the amount of input data that is buffered in the device driver during record operations. Applications that have particular requirements for low latency should set the value appropriately. Note however that smaller input buffer sizes may result in higher system overhead. The value of this field is specified in bytes and drivers will

constrain it to be a multiple of the current sample frame size. Some drivers may place other requirements on the value of this field. Refer to the audio device-specific manual page for more details. If an application changes the format of the audio device and does not modify the *record.buffer_size* field, the device driver may use a default value to compensate for the new data rate. Therefore, if an application is going to modify this field, it should modify it during or after the format change itself, not before. When changing the *record.buffer_size* parameters, the input stream should be paused and flushed before the change, and resumed afterward. Otherwise, subsequent reads may return samples in the old format followed by samples in the new format. This is particularly important when new parameters result in a changed sample size. If you change the *record.buffer_size* for the first packet, this protocol must be followed or the first buffer will be the default buffer size for the device, followed by packets of the requested change size.

The *record.buffer_size* field may be modified only on the `/dev/audio` device by processes that have it opened for reading.

The *play.buffer_size* field is currently not supported.

The audio data format is indicated by the *sample_rate, channels, precision* and encoding fields. The values of these fields correspond to the descriptions in the `AUDIO FORMATS` section of this man page. Refer to the audio device-specific manual pages for a list of supported data format combinations.

The data format fields can be modified only on the `/dev/audio` device.

If the parameter changes requested by an `AUDIO_SETINFO` ioctl cannot all be accommodated, `ioctl()` returns with `errno` set to `EINVAL` and no changes are made to the device state.

Streamio IOCTLS

All of the `streamio(7I)` `ioctl` commands may be issued for the `/dev/audio` device. Because the `/dev/audioctl` device has its own STREAMS queues, most of these commands neither modify nor report the state of `/dev/audio` if issued for the `/dev/audioctl` device. The `I_SETSIG` `ioctl` may be issued for `/dev/audioctl` to enable the notification of audio status changes, as described above.

Audio IOCTLS

The audio device additionally supports the following `ioctl` commands:

`AUDIO_DRAIN`
: The argument is ignored. This command suspends the calling process until the output STREAMS queue is empty and all queued samples have been played, or until a signal is delivered to the calling process. It may not be issued for the `/dev/audioctldevice`. An implicit `AUDIO_DRAIN` is performed on the final `close()` of `/dev/audio`.

`AUDIO_GETDEV`
: The argument is a pointer to an `audio_device_t` structure. This command may be issued for either `/dev/audio` or `/dev/audioctl`. The returned value in the name field will be a string that will identify the current `/dev/audio` hardware device, the value in version will be a string indicating the current version of the hardware, and `config` will be a

device-specific string identifying the properties of the audio stream associated with that file descriptor. Refer to the audio device-specific manual pages to determine the actual strings returned by the device driver.

`AUDIO_GETINFO` The argument is a pointer to an `audio_info_t` structure. This command may be issued for either `/dev/audio` or `/dev/audioctl`. The current state of the `/dev/audio` device is returned in the structure.

Values return pertain to a logical view of the device as seen by and private to the process, and do not necessarily reflect the actual hardware device itself.

`AUDIO_SETINFO` The argument is a pointer to an `audio_info_t` structure. This command may be issued for either the `/dev/audio` or the `/dev/audioctl` device with some restrictions. This command configures the audio device according to the supplied structure and overwrites the existing structure with the new state of the device. Note: The *play.samples*, *record.samples*, *play.error*, *record.error*, and *play.eof* fields are modified to reflect the state of the device when the `AUDIO_SETINFO` is issued. This allows programs to automatically modify these fields while retrieving the previous value.

As with `AUDIO_SETINFO`, the settings managed by this ioctl deal with a logical view of the device which is private to the process, and don't necessarily have any impact on the hardware device itself.

Certain fields in the audio information structure, such as the pause flags, are treated as read-only when `/dev/audio` is not open with the corresponding access permission. Other fields, such as the gain levels and encoding information, may have a restricted set of acceptable values. Applications that attempt to modify such fields should check the returned values to be sure that the corresponding change took effect. The *sample_rate*, *channels*, *precision*, and *encoding* fields treated as read-only for `/dev/audioctl`, so that applications can be guaranteed that the existing audio format will stay in place until they relinquish the audio device. `AUDIO_SETINFO` will return `EINVAL` when the desired configuration is not possible, or `EBUSY` when another process has control of the audio device.

All of the logical device state is reset when the corresponding I/O stream of `/dev/audio` is closed.

The `audio_info_t` structure may be initialized through the use of the `AUDIO_INITINFO` macro. This macro sets all fields in the structure to values that are ignored by the `AUDIO_SETINFO` command. For instance, the following code switches the output port from the built-in speaker to the headphone jack without modifying any other audio parameters:

```
audio_info_t info;
AUDIO_INITINFO();
```

```
info.play.port = AUDIO_HEADPHONE;
err = ioctl(audio_fd, AUDIO_SETINFO, );
```

This technique eliminates problems associated with using a sequence of `AUDIO_GETINFO` followed by `AUDIO_SETINFO`.

Errors An `open()` will fail if:

`EBUSY` The requested play or record access is busy and either the `O_NDELAY` or `O_NONBLOCK` flag was set in the `open()` request.

`EINTR` The requested play or record access is busy and a signal interrupted the `open()` request.

An `ioctl()` will fail if:

`EINVAL` The parameter changes requested in the `AUDIO_SETINFO()` ioctl are invalid or are not supported by the device.

Files The physical audio device names are system dependent and are rarely used by programmers. Programmers should use the following generic device names:

`/dev/audio`	Symbolic link to the system's primary audio device
`/dev/audioctl`	Symbolic link to the control device for `/dev/audio`
`/dev/sound/0`	First audio device in the system
`/dev/sound/0ctl`	Audio control device for `/dev/sound/0`
`/usr/share/audio/samples`	Audio files

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86
Availability	SUNWcsu, SUNWaudd, SUNWaudh
Stability Level	Obsolete Uncommitted

See Also `close(2)`, `fcntl(2)`, `ioctl(2)`, `open(2)`, `poll(2)`, `read(2)`, `write(2)`, `attributes(5)`, `dsp(7I)`, `streamio(7I)`

Bugs Due to a feature of the STREAMS implementation, programs that are terminated or exit without closing the audio device may hang for a short period while audio output drains. In general, programs that produce audio output should catch the `SIGINT` signal and flush the output stream before exiting.

Name audio810 – Intel ICH series, nVidia nForce series and AMD 8111 audio core support

Description The `audio810` driver provides support for AC 97 audio controllers embedded in Intel ICH, nVidia nForce, and AMD 8111 chips.

Files

`/kernel/drv/audio810`	32-bit kernel driver module
`/kernel/drv/amd64/audio810`	64-bit x86 kernel driver module
`/kernel/drv/audio810.conf`	`audio810` driver configuration file

Attributes See `attributes(5)` for a descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	PC-based systems
Availability	SUNWad810
Interface Stability	Uncommitted

See Also `ioctl(2)`, `attributes(5)`, `audio(7I)`, `mixer(7I)`, `streamio(7I)`

AMD-8111 HyperTransport I/O Hub Data Sheet — Advanced Micro Devices Inc.

ALC655 Specification — Realtek Inc.

Notes Some laptops (including Sony VAIO, among others), have their on-board amplifier powered down by default, meaning that audio is suppressed even if hardware and the `audio810` driver are working normally. To correct this, set the `ac97-invert-amp=1` property in the `/kernel/drv/audio810.conf` to power-up the amplifier.

Diagnostics In addition to being logged, the following messages may appear on the system console:

`play-interrupts too low` `record-interrupts too low`	The interrupt rate in `audio810.conf` is set too low. It has been reset to the rate specified in the message. Update `audio810.conf` to a higher interrupt rate.
`play-interrupts too high` `record-interrupts too high`	The interrupt rate set in`audio810.conf` is set too high.It has been reset to the rate specified in the message. Update `audio810.conf`to a lower interrupt rate.

Name audiocmi – C-Media 8738, 8768, and 8338 driver support

Description The `audiocmi` driver provides support for the C-Media 8738, 8768, and 8338 audio controllers. These are found on some motherboards and some add-in PCI cards.

Files

`/kernel/drv/audiocmi`	32-bit kernel driver module
`/kernel/drv/amd64/audiocmi`	64-bit x86 kernel driver module
`/kernel/drv/sparcv9/audiocmi`	64-bit SPARC kernel driver module
`/kernel/drv/audiocmi.conf`	Driver configuration file

Attributes See `attributes(5)` for a descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	PC-based system
Availability	SUNWaudiocmi

See Also `ioctl(2)`, `attributes(5)`, `audio(7I)`, `mixer(7I)`, `streamio(7I)`

Name audiocs – Crystal Semiconductor 4231 Audio driver

Description The `audiocs` driver supports the Crystal Semiconductor 4231 Codec to implement the audio device interface.

The `audiocs` device provides support for the internal speaker, headphone, line out, line in, microphone, and, on some platforms, internal CD-ROM audio in.

Errors `audiocs` errors are described in the `audio(7I)` manual page.

Files

`/kernel/drv/sparcv9/audiocs`	64-bit `audiocs` driver
`/kernel/drv/audiocs.conf`	`audiocs` driver configuration file

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC
Availability	SUNWaudd
Stability level	Uncommitted

See Also `ioctl(2)`, `attributes(5)`, `audio(7I)`, `mixer(7I)`, `streamio(7I)`

Crystal Semiconductor, Inc. CS4231 Data Sheet

Diagnostics In addition to being logged, the following messages can appear on the system console:

`play-interrupts too low`
`record-interrupts too low` — The interrupt rate specified in `audiocs.conf` is set too low. It is being reset to the rate specified in the message. Update `audiocs.conf` to a higher interrupt rate.

`play-interrupts too high`
`record-interrupts too high` — The interrupt rate specified in `audiocs.conf` is set too high. It is being reset to the rate specified in the message. Update `audiocs.conf` to a lower interrupt rate.

Name audioemu10k – Creative EMU10K audio device support

Description The `audioemu10k` driver provides support for the Creative EMU 10K1 and 10K2 family of audio devices. These are typically marketed under the Audigy or Sound Blaster Live! brands.

This device driver is capable of 5.1 or 7.1 surround sound and SPDIF playback and record, depending on the capabilities of the individual device.

Files

`/kernel/drv/audioemu10k`	32-bit kernel driver module
`/kernel/drv/amd64/audioemu10k`	64-bit kernel driver module

Attributes See `attributes(5)` for a descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWaudioemu10k

See Also `ioctl(2)`, `attributes(5)`, `audio(7I)`, `mixer(7I)`, `streamio(7I)`

Name audioens – Ensoniq ESS 1371 and ESS 1373 audio driver

Description The `audioens` driver provides support for the Ensoniq ESS1371, ESS1373, and Creative 5880 AC'97 devices. These devices are commonly known by several different names, including the Sound Blaster PCI128 and AudioPCI '97.

Files

`/kernel/drv/audioens`	32-bit kernel module
`/kernel/drv/amd64/audioens`	64-bit x86 kernel module
`/kernel/drv/sparcv9/audioens`	64-bit SPARC kernel module

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86
Availability	SUNWaudd
Interface Stability	Uncommitted

See Also `attributes(5)`, `audio(7I)`, `dsp(7I)`, `mixer(7I)`

Name audiohd – Intel High Definition Audio Controller support

Description The `audiohd` driver provides support for the generic codec chips which are compatible with the Intel High-Definition Audio Controller 1.0 specification.

Files

`/kernel/drv/audiohd.conf`	`audiohd` driver configuration file
`/kernel/drv/audiohd`	32-bit x86 kernel driver module
`/kernel/drv/amd64/audiohd`	64-bit x86 kernel driver module

Attributes See `attributes(5)` for a descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	PC-based system
Availability	SUNWaudiohd
Stability level	Uncommitted

See Also `ioctl(2)`, `attributes(5)`, `audio(7I)`, `mixer(7I)`, `streamio(7I)`

Intel High-Definition Audio Specification 1.0. - Intel Corporation

ALC880 Specification — Realtek Inc.

Diagnostics In addition to being logged, the following messages can appear on the system console:

`play-interrupts too low`
`record-interrupts too low` — The interrupt rate in `audiohd.conf` is set too low. It has been reset to the rate specified in the message. Update `audiohd.conf` to a higher interrupt rate.

`redcord-interrupts too low`
`record-interrupts too high` — The interrupt rate in `audiohd.conf` is set too low. It has been reset to the rate specified in the message. Update `audiohd.conf` to a higher interrupt rate

Name audioixp – ATI IXP400 south bridge audio digital controller interface

Description The `audioixp` device uses the IXP400 south bridge audio digital controller and a AC–97 Codec to implement the audio device interface.

The `audioixp` device provides support for the internal speaker, headphone, line out, line in, and microphone.

Files

`/kernel/drv/audioixp.conf`	Driver configuration file
`/kernel/drv/audioixp`	32-bit kernel driver module
`/kernel/drv/amd64/audioixp`	64-bit kernel driver module

Attributes See `attributes(5)` for a descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	PC-based system
Availability	SUNWadixp
Stability level	Uncommitted

See Also `ioctl(2)`, `attributes(5)`, `audio(7I)`, `mixer(7I)`, `streamio(7I)`

ATI IXP400 South Bridge Data Sheet

Diagnostics In addition to being logged, the following messages may appear on the system console:

`play-interrupts too low`
`record-interrupts too low` The interrupt rate in `audioixp.conf` is set too low. It has been reset to the rate specified in the message. Update `audioixp.conf` to a higher interrupt rate.

`play-interrupts too high`
`record-interrupts too high` The interrupt rate set in `audioixp.conf` is set too high. It has been reset to the rate specified in the message. Update `audioixp.conf` to a lower interrupt rate.

Name audiols – Creative Audigy LS audio device support

Description The `audiols` driver provides support for the Creative Audigy LS audio device.

There are numerous devices marketed under the Audigy brand by Creative, but only Audigy LS devices are supported by this driver.

This device is capable of 5.1 surround sound.

Files

`/kernel/drv/audiols`	32-bit kernel driver module
`/kernel/drv/amd64/audiols`	64-bit kernel driver module

Attributes See `attributes(5)` for a descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWaudiols

See Also `ioctl(2)`, `attributes(5)`, `audio(7I)`, `mixer(7I)`, `streamio(7I)`

Name audiop16x – Creative Sound Blaster Live! OEM support

Description The `audiop16x` driver provides support for the Creative Sound Blaster Live! products based on the P16X device. These chips are also known as the `EMU10K1X` device, not to be confused with the `EMU10K1`.

Add-in boards known to work with this driver are Sound Blaster Live! cards with model numbers SB0200 or SB0213.

This device is capable of 5.1 surround sound.

Files

`/kernel/drv/audiop16x`	32-bit kernel driver module
`/kernel/drv/amd64/audiop16x`	64-bit x86 kernel driver module
`/kernel/drv/sparcv9/audiop16x`	64-bit SPARC kernel driver module

Attributes See `attributes(5)` for a descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86, SPARC
Availability	SUNWaudiop16x

See Also `ioctl(2)`, `attributes(5)`, `audio(7I)`, `mixer(7I)`, `streamio(7I)`

Name audiopci – Ensoniq 1370 driver support

Description The `audiopci` driver provides support for the Ensoniq 1370 audio controller. Ensoniq 1370 chips are found on add-in PCI cards commonly identified as Audio PCI and SoundBlaster PCI.

Files

`/kernel/drv/audiopci`	32-bit kernel driver module
`/kernel/drv/amd64/audiopci`	64-bit x86 kernel driver module
`/kernel/drv/amd64/audiopci`	64-bit SPARC kernel driver module
`/kernel/drv/audiopci.conf`	Driver configuration file

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	PCI-based systems
Availability	SUNWaudd

See Also `ioctl(2)`, `attributes(5)`, `audio(7I)`, `mixer(7I)`, `streamio(7I)`

Creative Technology Ltd ES1370 Specification

http://www.sun.com

Name audiosolo – ESS Solo-1 audio device support

Description The `audiosolo` driver provides support for the ESS Solo-1 audio device. This device is found on certain motherboards and discrete audio cards. It supports 16-bit 48 kHZ stereo playback and capture.

Files

`/kernel/drv/audiosolo`	32-bit kernel driver module
`/kernel/drv/amd64/audiosolo`	64-bit kernel driver module

Attributes See `attributes(5)` for a descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWaudiosolo

See Also `ioctl(2)`, `attributes(5)`, `audio(7I)`, `mixer(7I)`, `streamio(7I)`

Name audiots – Acer Laboratories Inc. M5451 audio processor interface

Description The `audiots` device uses the ALI M5451 audio processor and an AC-97 Codec to implement the audio device interface.

The `audiots` device provides support for the internal speaker, headphone, line out, line in, and microphone.

Files

`/kernel/drv/sparcv9/audiots`	64-bit `audiots` driver
`/kernel/drv/audiots.conf`	`audiots` driver configuration file

Attributes See `attributes(5)` for a descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC
Availability	SUNWaudd
Stability level	Uncommitted

See Also `ioctl(2)`, `attributes(5)`, `audio(7I)`, `mixer(7I)`, `streamio(7I)`

Acer Laboratories Inc. M5451 PCI Audio Processor Technical Specification

Diagnostics In addition to being logged, the following messages can appear on the system console:

`play-interrupts too low` `record-interrupts too low`	The interrupt rate in `audiots.conf` is set too low. It has been reset to the rate specified in the message. Update `audiots.conf` to a higher interrupt rate.
`play-interrupts too high` `record-interrupts too high`	The interrupt rate set in `audiots.conf` is set too high. It has been reset to the rate specified in the message. Update `audiots.conf` to a lower interrupt rate.

Name audiovia823x – VIA VT8233, VT8235, and VT8237) support

Description The `audiovia823`*x* driver provides support for the VIA VT8233, VT8235, and VT8237 AC'97 devices found on motherboards with certain VIA chip sets.

Files

`/kernel/drv/audiovia823x`	32-bit x86 kernel module
`/kernel/drv/amd64/audiovia823x`	64-bit x86 kernel module

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWvia823x
Interface Stability	Committed

See Also `attributes(5)`, `audio(7I)`, `dsp(7I)`, `mixer(7I)`

Name audiovia97 – Via 82C686 audio device support

Description The `audiovia97` driver provides support for the integrated audio device found in the Via 82C686 southbridge chipset, found on certain motherboards.

Files `/kernel/drv/audiovia97` 32-bit kernel driver module

Attributes See `attributes(5)` for a descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	Via 82C686-based systems
Availability	SUNWaudiovia97

See Also `ioctl(2)`, `attributes(5)`, `audio(7I)`, `mixer(7I)`, `streamio(7I)`

Name av1394 – 1394 audio/video driver

Synopsis `unit@GUID`

Description The `av1394` driver implements `iec61883(7I)` interfaces for IEEE 1394 compliant devices.

Asynchronous Transactions

The driver allows applications to act as FCP controllers, but not FCP targets. Only `IEC61883_FCP_CMD` requests can be sent with `write(2)`. Only `IEC61883_FCP_RESP` requests can be received with `read(2)`.

Isochronous Transactions

When the read/write method of is used for transmit, the driver is capable of auto-detecting and transmitting SD-DVCR 525/60 and 625/50 streams. See `iec61883(7I)` for details.

Files

`/dev/av/N/async`	device node for asynchronous data
`/dev/av/N/isoch`	device node for isochronous data
`kernel/drv/sparcv9/av1394`	64-bit ELF kernel module
`kernel/drv/av1394`	32-bit ELF kernel module
`kernel/drv/amd64/av1394`	64-bit ELF kernel module

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	All
Interface Stability	Committed

See Also `read(2)`, `write(2)`, `attributes(5)`, `hci1394(7D)`, `iec61883(7I)`

IEEE Std 1394-1995 Standard for a High Performance Serial Bus

IEC 61883 Consumer audio/video equipment - Digital interface

Name bbc_beep – Platform-dependent Beep driver for BBC-based hardware.

Synopsis

```
beep@unit-address
```

Description The `bbc_beep` driver generates beeps on platforms (including Sun Blade 1000) that use BBC-based registers and USB keyboards. When the `KIOCCMD` ioctl is issued to the USB keyboard module (see `usbkbm`(7M)) with command `KBD_CMD_BELL/KBD_CMD_NOBELL`, `usbkbm`(7M) passes the request to the `bbc_beep` driver to turn the beep on and off, respectively.

Files

`/platform/sun4u/kernel/drv/sparcv9/bbc_beep`
64–bit ELF kernel driver

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	BBC-based SPARC
Availability	SUNWcarx.u

See Also `kbd`(1), `attributes`(5), `grbeep`(7d), `kb`(7M), `usbkbm`(7M)

Writing Device Drivers

Diagnostics None

Name bcm_sata – Broadcom HT1000 SATA controller driver

Synopsis `sata@`*unit-address*

Description The `bcm_sata` driver is a SATA HBA driver that supports Broadcom HT1000 SATA HBA controllers.

HT1000 SATA controllers are compliant with the Serial ATA 1.0 specification and SATA II Phase 1.0 specification (the extension to the SATA 1.0 specification). These HT1000 controllers support standard SATA features including SATA-II disks, NCQ, hotplug, ATAPI devices and port multiplier.

The driver does not currently support NCQ and port multiplier features.

Configuration The `bcm_sata` module contains no user configurable parameters.

Files

`/kernel/drv/bcm_sata`	32-bit ELF kernel module (x86)
`/kernel/drv/amd64/bcm_sata`	64-bit ELF kernel module (x86)

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWbcmsata

See Also `cfgadm(1M)`, `cfgadm_sata(1M)`, `prtconf(1M)`, `attributes(5)`, `sata(7D)`, `sd(7D)`

Writing Device Drivers

Name bfe – Device driver for Broadcom BCM4401 100Base-T NIC

Description The `bfe` Fast Ethernet driver is GLD-based and supports the Broadcom BCM4401 100Base-T NIC adapters :`pci14e4,170c Broadcom BCM4401 100Base-T.`.

The `bfe` driver supports IEEE 802.3 auto-negotiation, flow control and VLAN tagging.

Configuration The default configuration is auto-negotiation with bidirectional flow control. The advertised capabilities for auto-negotiation are based on the capabilities of the `PHY`.

You can set the capabilities advertised by the `bfe` controlled device using `dladm`(1M). The driver supports only those parameters which begin with en (enabled) in the parameters listed by the command `dladm`(1M). Each of these boolean parameters determines if the device advertises that mode of operation when the hardware supports it.

Files

`/dev/bfe`	Special character device
`/kernel/drv/bfe`	32-bit device driver (x86)
`/kernel/drv/amd64/bfe`	64-bit device driver (x86)

Attributes See `attributes`(5) for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86

See Also `dladm`(1M), `netstat`(1M), `driver.conf`(4), `attributes`(5), `ieee802.3`(5), `dlpi`(7P), `streamio`(7I)

Writing Device Drivers

STREAMS Programmer's Guide

Network Interface Guide

IEEE 802.3ae Specification - 2002

Name bge – SUNW,bge Gigabit Ethernet driver for Broadcom BCM57xx

Synopsis `/dev/bge*`

Description The `bge` Gigabit Ethernet driver is a multi-threaded, loadable, clonable, GLD-based STREAMS driver supporting the Data Link Provider Interface, `dlpi(7P)`, on Broadcom BCM57xx (BCM5700/5701/5703/5704/5705/5705M/5714/5721/5751/5751M/5782/5788 on x86) Gigabit Ethernet controllers fitted to the system motherboard. With the exception of BCM5700/BCM5701/BCM5704S, these devices incorporate both MAC and PHY functions and provide three-speed (copper) Ethernet operation on the RJ-45 connectors. (BCM5700/BCM5701/BCM5704S do not have a PHY integrated into the MAC chipset.)

The `bge` driver functions include controller initialization, frame transmit and receive, promiscuous and multicast support, and error recovery and reporting.

The `bge` driver and hardware support auto-negotiation, a protocol specified by the 1000 Base-T standard. Auto-negotiation allows each device to advertise its capabilities and discover those of its peer (link partner). The highest common denominator supported by both link partners is automatically selected, yielding the greatest available throughput, while requiring no manual configuration. The `bge` driver also allows you to configure the advertised capabilities to less than the maximum (where the full speed of the interface is not required), or to force a specific mode of operation, irrespective of the link partner's advertised capabilities.

Application Programming Interface The cloning character-special device, `/dev/bge`, is used to access all BCM57xx devices ((BCM5700/5701/5703/5704, 5705/5714/5721/5751/5751M/5782 on x86) fitted to the system motherboard.

The `bge` driver is managed by the `dladm(1M)` command line utility, which allows VLANs to be defined on top of bge instances and for `bge` instances to be aggregated. See `dladm(1M)` for more details.

You must send an explicit DL_ATTACH_REQ message to associate the opened stream with a particular device (PPA). The PPA ID is interpreted as an unsigned integer data type and indicates the corresponding device instance (unit) number. The driver returns an error (DL_ERROR_ACK) if the PPA field value does not correspond to a valid device instance number for the system. The device is initialized on first attach and de-initialized (stopped) at last detach.

The values returned by the driver in the DL_INFO_ACK primitive in response to a DL_INFO_REQ are:

- Maximum SDU (default 1500).
- Minimum SDU (default 0).
- DLSAP address length is 8.
- MAC type is `DL_ETHER`.
- SAP length value is *-2*, meaning the physical address component is followed immediately by a 2-byte SAP component within the `DLSAP` address.

- Broadcast address value is the Ethernet/IEEE broadcast address (FF:FF:FF:FF:FF:FF).

Once in the DL_ATTACHED state, you must send a DL_BIND_REQ to associate a particular Service Access Point (SAP) with the stream.

Configuration By default, the `bge` driver performs auto-negotiation to select the link speed and mode. Link speed and mode can be any one of the following, (as described in the *IEEE803.2* standard):

- 1000 Mbps, full-duplex
- 1000 Mbps, half-duplex
- 100 Mbps, full-duplex
- 100 Mbps, half-duplex
- 10 Mbps, full-duplex
- 10 Mbps, half-duplex

The auto-negotiation protocol automatically selects:

- Speed (1000 Mbps, 100 Mbps, or 10 Mbps)
- Operation mode (full-duplex or half-duplex)

as the highest common denominator supported by both link partners. Because the `bge` device supports all modes, the effect is to select the highest throughput mode supported by the other device.

Alternatively, you can set the capabilities advertised by the `bge` device using `dladm(1M)`. The driver supports a number of parameters whose names begin with *en_* (see below). Each of these parameters contains a boolean value that determines whether the device advertises that mode of operation. If `en_autoneg_cap` is set to 0, the driver forces the mode of operation selected by the first non-zero parameter in priority order as listed below:

```
(highest priority/greatest throughput)
      en_1000fdx_cap        1000Mbps full duplex
      en_1000hdx_cap        1000Mpbs half duplex
      en_100fdx_cap         100Mpbs full duplex
      en_100hdx_cap         100Mpbs half duplex
      en_10fdx_cap          10Mpbs full duplex
      en_10hdx_cap          10Mpbs half duplex
                      (lowest priority/least throughput)
```

For example, to prevent the device 'bge2' from advertising gigabit capabilities, enter (as super-user):

```
# dladm set-linkprop -p enable_1000hdx_cap=0  bge2
# dladm set-linkprop -p enable_1000fdx_cap=0 bge2
```

All capabilities default to enabled. Note that changing any capability parameter causes the link to go down while the link partners renegotiate the link speed/duplex using the newly changed capabilities.

The current settings of the parameters may be found using dladm show-ether. In addition, the driver exports the current state, speed, duplex setting, and working mode of the link via kstat parameters (these are read only and may not be changed). For example, to check link state of device `bge0`:

```
# dladm show-ether -x bge0
LINK        PTYPE    STATE    AUTO  SPEED-DUPLEX            PAUSE
bge0        current  up       yes   1G-f                    bi
--          capable  --       yes   1G-fh,100M-fh,10M-fh    bi
--          adv      --       yes   1G-fh                   bi
--          peeradv  --       yes   1G-f                    bi
```

The output above indicates that the link is up and running at 1Gbps full-duplex with its rx/tx direction pause capability.

To extract link state information for the same link using kstat:

```
# kstat bge:0:mac:link_state
module: bge                 instance: 0
name:   mac                 class:    net
link_state
```

The default MTU is 1500. To enable Jumbo Frames support, you can configure the bge driver by defining the default_mtu property via `dladm(1M)` or in `driver.conf(4)` to greater than 1500 bytes (for example: default_mtu=9000). Note that the largest jumbo size supported by bge is 9000 bytes. Additionally, not all bge-derived devices currently support Jumbo Frames. The following devices support Jumbo Frames up to 9KB: BCM5700, 5701, 5702, 5703C, 5703S, 5704C, 5704S, 5714C, 5714S, 5715C and 5715S. Other devices currently do not support Jumbo Frames.

Files

`/kernel/drv/bge*`	32–bit ELF kernel module. (x86)
`/kernel/drv/amd64/bge`	64–bit ELF kernel module (x86).
`/kernel/drv/sparcv9/bge`	64–bit ELF kernel module (SPARC).

Attributes

See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86

See Also

`dladm(1M)`, `driver.conf(4)`, `attributes(5)`, `streamio(7I)`, `dlpi(7P)`

Writing Device Drivers

STREAMS Programming Guide

Network Interfaces Programmer's Guide

Name blk2scsa – SCSA block device emulation

Description The `blk2scsa` module provides support services for generic block devices so that they appear to the system as devices on a virtual SCSI bus, thus allowing them to be serviced by the `sd(7D)` SCSI disk driver. The `blk2scsa` device supports the SCSI-2 command set for Direct Access Devices. The `blk2scsa` device supports multiple LUNs per physical device and creates a separate child device for each LUN. All child nodes attach to `sd(7D)`.

Device Special Files Disk block special file names are located in `/dev/dsk`. Raw file names are located in `/dev/rdsk`. See `sd(7D)`.

ioctls See `dkio(7I)`

Errors See `sd(7D)`.

Files Device special files for the storage device are created in the same way as those for a SCSI disk. See `sd(7D)` for more information.

/dev/dsk/*cntndnsn*	Block files for disks.
/dev/rdsk/ct*n*d*n*s*n*	Raw files for disks.
`/kernel/misc/blk2scsa`	32-bit ELF kernel module (x86).
`/kernel/misc/amd64/blk2scsa`	64-bit ELF kernel module (x86).
`/kernel/misc/sparcv9/blk2scsa`	64-bit ELF kernel module (SPARC).

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86
Availability	SUNWckr

See Also `eject(1)`, `rmformat(1)`, `rmmount(1)`, `cfgadm_scsi(1M)`, `fdisk(1M)`, `mount(1M)`, `umount(1M)`, `scsi(4)`, `vfstab(4)`, `attributes(5)`, `sd(7D)`, `dkio(7I)`, `pcfs(7FS)`

Name bnx – Broadcom NetXtreme II Gigabit Ethernet Device Driver

Synopsis `/dev/bnx`

Description The `bnx` Gigabit Ethernet driver is a multi-threaded, loadable, clonable, GLD v3-based STREAMS driver supporting the Data Link Provider Interface, `dlpi(7P)`, over Broadcom NetXtreme II Ethernet controllers, including the BCM5706, BCM5708 and BCM5709 controllers. Driver functions include controller initialization, frame transmit and receive, promiscuous and multicast support and error recovery and reporting.

Application Programming Interface The cloning, character-special device `/dev/bnx` is used to access all Broadcom NetXtreme II Ethernet devices installed within the system.

The `bnx` driver is dependent on `/kernel/misc/mac`, a loadable kernel module that provides the `bnx` driver with the DLPI and STREAMS functionality required of a LAN driver.

The values returned by the driver in the `DL_INFO_ACK` primitive in response to the `DL_INFO_REQ` are:

- Maximum `SDU` (with jumbo frame) is 9000.
- Minimum `SDU` is `0`. The driver pads to 60-byte minimum packet size.
- `DSLAP` address length is `8` bytes.
- `MAC` type is `DL_ETHER`.
- `SAP` length value is -2, meaning the physical address component is followed immediately by a 2-byte sap component within the DLSAP address.
- Version is `DL_VERSION_2`.
- Broadcast address value is Ethernet/IEEE broadcast address (`FF:FF:FF:FF:FF:FF`).

Configuration By default, the `bnx` driver performs auto-negotiation to select the link speed and mode. Link speed and mode can be any of the following:

2500 Mbps, full-duplex (fiber physical interface controller only)

1000 Mbps, full-duplex

100 Mbps, full-duplex

100 Mbps, half-duplex

10 Mbps, full-duplex

10 Mbps, half-duplex

To customize the driver parameters, edit the `/kernel/drv/bnx.conf` file. The driver properties are:

`adv_*`
The adv parameters are advertised to the link partner and include:

```
adv_autoneg_cap
adv_pause_cap
adv_2500fdx_cap
adv_1000fdx_cap
adv_1000hdx_cap
adv_100fdx_cap
adv_100hdx_cap
adv_10fdx_cap
adv_10hdx_cap
```

`transfer_speed`
The driver attempts to auto-negotiate but is restricted to the specified speed. Duplex mode is determined through auto-negotiation.

`speed`
`full-duplex`
Forces speed and duplex mode to a fixed value. This value take precedence over others.

`speed`
Configures link (or instance) to a designated speed. By default, `AutoNegotiate` (`0`) is set. The setup is based on the following values:

`0`	AutoNegotiate.
`10`	10 Mbps speed mode (Copper only).
`100`	100 Mbps speed mode (Copper only).
1000	1000 Mbps speed mode (Copper and fiber).
`2500`	2500 Mbps speed mode (Fiber only).

`Flow`
Configures flow control parameters of a link. The setup is based on the following values:

`0`	`Tx` and `Rx` flow control are disabled.
`1`	`Tx` flow control is enabled. Pause frames are sent if resource is low, but device does not process `Rx` Pause Frame.
`2`	Only `Rx` flow control is enabled. If device receives Pause Frame, it stops sending.
`3`	`Rx` and `TX` flow control are enabled. Pause frames are sent if resource is low. If device receives Pause Frame, it stops sending.
`4`	Advertise `Rx` and `TX` flow control are enabled and negotiating with link partner. If link AutoNegotiate is not enabled, `Tx` and `Rx` Flow Control are disabled.

`Jumbo`
Configures Jumbo Frame link feature. Valid range for this parameter is `0` to `3800`. If value configured is less then `1500`, Jumbo Frame feature is disabled.

`RxBufs`
Configures number of `Rx` packet descriptor. The valid value is `32` to `1024`. More system memory resource is used for larger number of `Rx` Packet Descriptors. Default value is `500`.

`RxTicks`
Configures number of `Rx` Host Coalescing Ticks in microseconds. This determines the maximum time interval in which the device generates an interrupt if one or more frames are received. The default value is `25`.

`Coalesce`
Configures number of `Tx/Rx` Maximum Coalesced Frames parameters. This determines the maximum number of buffer descriptors the device processes before it generates an interrupt. The default value is `16`.

`TxTicks`
Configures number of `Tx` Host Coalescing Ticks in microseconds. This determines the maximum time interval in which the device generates an interrupt if one or more frames are sent. The default value is 45.

`TxMaxCoalescedFrames`
Configures number of `Tx` Maximum Coalesced Frames parameters. This determines the maximum number of `Tx` buffer descriptors the device processes before it generates an interrupt. The default value is `80`.

`RxTicksInt`
Configures number of `Rx` Host Coalescing Ticks in microseconds during interrupt. This determines the maximum time interval in which the device generates interrupt if one or more frames are received during interrupt handling. The default value is `15`.

`TxTicksInt`
Configures number of Tx Host Coalescing Ticks in microseconds during interrupt. This determines the maximum time interval in which the device generates an interrupt if one or more frames are received during interrupt handling. The default value is 15.

`StatsTicks`
Configures how often adapter statistics are DMA'd to host memory in microsecond. Default is `1000000`.

Configuring with ndd(1M)

You can also perform configuration tasks using ndd(1M). For example, to prevent the device `bnx1` from advertising gigabit capabilities, do the following as super-user:

```
# ndd -set /dev/bnx1 adv_1000fdx_cap 0
```

All capabilities default to enabled and that changing any parameter causes the link to go down while the link partners renegotiate the link speed/duplex. To view current parameters, use

`ndd -get`. In addition, the driver exports the current state, speed, duplex setting and working mode of the link by way of the `ndd` parameters, which are read only and cannot be changed. For example, to check the state of device `bnx0`:

```
# ndd -get /dev/bnx0 link_status
1
# ndd -get /dev/bnx0 link_speed
100
# ndd -get /dev/bnx0 link_duplex
2
```

The output above indicates that the link is up and running at 100Mbps full-duplex.

Files

`/dev/bnx`	Special character device
`/kernel/drv/bnx`	32-bit ELF kernel module (x86)
`/kernel/drv/amd64/bnx`	64-bit ELF Kernel module (x86)
`/kernel/drv/bnx.conf`	Driver configuration file

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	BRCMbnx
Architecture	x86
Interface Stability	See below.

The `bnx` driver is Committed. The `/kernel/drv/bnx.conf` configuration file is Uncommitted.

See Also `dladm(1M)`, `ndd(1M)`, `attributes(5)`, `streamio(7I)`, `dlpi(7P)`

Writing Device Drivers

STREAMS Programming Guide

Network Interfaces Programmer's Guide

Name bnxe – Broadcom NetXtreme II 10 Gigabit Ethernet Device Driver

Synopsis `/dev/bnxe*`

Description The `bnxe` Ethernet driver is a multi-threaded, loadable, clonable, GLDv3-based driver supporting the Data Link Provider Interface, `dlpi(7P)`, over Broadcom NetXtreme II 10 Gigabit Ethernet controllers. Multiple NetXtreme II controllers installed within the system are supported by the driver.

The `bnxe` driver provides basic support for the NetXtreme II 10 Gigabit line of devices. Functions include chip initialization, frame transit and receive, multicast and promiscuous support, and error recovery and reporting. NetXtreme II 10 Gigabit devices provide 10/100/1000/10000 Mbps networking interfaces for Copper and 1000/2500/10000 Mbps for Fiber physical interfaces.

The *IEEE 802.3* standard specifies an *auto-negotiation* protocol to automatically select the mode and speed of operation. The PHY device is capable of doing auto-negotiation with the remote-end of the link (Link Partner) and receives the capabilities from the remote end. It selects the Highest Common Denominator mode of operation. It also supports forced-mode of operation where the driver can select desired mode of operation.

Driver Configuration

There are two facilities by which the administrator can configure each NetXtreme II 10 Gigabit device in the system, the hardware configuration file and the NDD subsystem. The hardware configuration file is located at `/kernel/drv/bnxe.conf` and contains a list of options that are read when the driver is loaded. The NDD subsystem options are used as a way to modify the device's configuration at runtime. All changes made with the NDD subsystem are lost after device reset or system reboot. The remainder of this section discusses configuration options common to both facilities.

Link Speed and Duplex Parameters

The primary way link speed and duplex settings are configured is through the following list of options. `Non-zero` values turn them on, a value of zero disables them.

- `adv_autoneg_cap`
- `adv_10000fdx_cap`
- `adv_2500fdx_cap`
- `adv_1000fdx_cap`
- `adv_100fdx_cap`
- `adv_100hdx_cap`
- `adv_10fdx_cap`
- `adv_10hdx_cap`

When the `adv_autoneg_cap` option is set to a `non-zero` value, the remaining options control which capabilities we advertise to the link partner during auto-negotiation. When the `adv_autoneg_cap` option is set to `zero`, the driver walks down the list, from the highest speed / duplex option to the lowest, and use the first `non-zero` option as the speed / duplex setting to force the link with.

Flow Control Parameters

Flow control parameters configure the flow control properties of the physical link. The properties are configured through the following list of options. Like the link speed and duplex settings, `non-zero` values turn them on, a value of `zero` disables them.

`autoneg_flow` Controls whether or not the flow control properties are auto-negotiated or forced.

`txpause_cap` Controls whether or not `Tx` flow control can be configured.

`rxpause_cap` Controls whether or not `Rx` flow control can be configured.

If `Tx` flow control is enabled, pause frames are sent to the link partner when `Rx` resources are low. If `Rx` flow control is enabled, the hardware automatically stops transmitting if pause frames are received. How `Tx` and `Rx` flow control are enabled depends on how the driver is configured. When the `autoneg_flow` option is `non-zero`, the flow control capabilities become advertisement settings and the`auto-negotiation` process dictates what actual flow control settings are used. If the `autoneg_flow` option is zero, the flow control capabilities specified are still advertised if the link is auto-negotiated, but the actual flow control settings are forced to the specified settings.

Hardware Configuration File Options

`checksum` This parameter configures checksum calculation tasks to be offloaded to the hardware. If 0 then no checksums are offloaded. If 1 then IPv4 header checksums offloaded for Rx/Tx. If 2 then TCP/UDP/IPv4 header checksums are offloaded for Rx/Tx. The default is `2` (TCP/UDP/IPv4 checksums).

`mtu` This parameter controls the MTU (Message Tranfer Unit) size of the hardware. Egress Ethernet frames larger than this size is fragmented and ingress Ethernet frames larger than this size is dropped. The valid range is from `60` to `9216` (decimal). The default value is `1500`.

`rx_descs`
`tx_descs` These parameters control how many packets can be in-transit within the driver. The greater the number of packets that are allowed to be in-transit, the more memory the driver requires to operate. The valid range is `1` to `32767`. The default value is `1280`.

`rx_free_reclaim`
`tx_free_reclaim` These parameters control the threshold of freely available packet descriptors that are allowed to be free before reposting back to the hardware so they can be reused for future packets. The valid range is 0 to the value of `rx_descs` for `Rx` and `tx_descs` for `Tx`. A freely available packet descriptor is, for `Rx` a packet that has been received

and processing finished, and for Tx a packet that has already been sent. The default value is 1/16 of rx_descs for Rx and tx_descs for Tx.

interrupt_coalesce — This parameter gives the administrator the ability to allow the hardware to collect more network events before interrupting the host processor. Interrupt coalescing within the NetXtreme II 10 Gigabit hardware is quite aggressive resulting in great sustained throughput but low latency for interactive traffic. Setting to 1 turns it on and 0 off. Default is off.

rx_copy_threshold
tx_copy_threshold — These parameters control the packet size threshold, in number of bytes, when packets are double copied before processing. A value of 0 turns off double copies. For Tx a value of 1 means all packets are copied. For Rx a really large value, that is, greater than the mtu, that means all packets are copied. The default is 0 for both Rx and Tx which implies that no copying is ever performed.

NDD Subsystem Configuration Options

Hardware Capability Parameters

The following is a read-only list of capabilities the device is capable of supporting. A value of 1 means the capability is supported. A value of 0 means the capability is not supported.

- autoneg_cap
- 10000fdx_cap
- 2500fdx_cap
- 1000fdx_cap
- 1000fdx_cap
- 100hdx_cap
- 10fdx_cap
- 10hdx_cap
- 10hdx_cap
- txpause_cap

Hardware Capability Advertisement Parameters

The auto-negotiation advertisement parameters work exactly as described in the Hardware Capability Parameters section of this man page with the difference that they show what is actually being advertised

Miscellaneous Link Parameters

These parameters show the other phy options.

- autoneg_flow

Link Status Parameters

These parameters show the current status of the physical link.

`link_status` Shows if the link is up or down.

`link_speed` Shows the current speed of the link.

`link_duplex` Shows the current duplex setting of the link.

`link_txpause` Shows whether or not TX flow control is enabled.

`link_rxpause` Shows whether or not RX flow control is enabled.

Convenience Parameters

The following parameters display multiple settings at once. They are intended for convenience.

`hw_cap` Shows all the device hardware capabilities

`adv_cap` Shows all the capabilities we are advertising or have forced.

Debug

The following parameters are used to aid in debugging driver issues. These should be used in conjunction with `kstat` statistic diagnostics.

`debug` Shows the state of all the internal packet descriptor queues.

Files

`/dev/bnx[instance]`	`bnxe` character special device.
`/kernel/drv/bnxe.conf`	Configuration file of the `bnxe` driver.
`/kernel/drv/bnxe`	32-bit i386 driver binary
`/kernel/drv/amd64/bnxe`	64-bit i386 driver binary
`release.txt`	Revision history of the driver

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	BRCMbnxe
Architecture	x86

See Also

`ndd(1M)`, `attributes(5)`, `dlpi(7P)`, `gld(7D)`,

Broadcom NetXtreme II 10 Gigabit Adapter Driver Installation Notes

Writing Device Drivers

STREAMS Programming Guide

Network Interfaces Programmer's Guide

Name bpf – Berkeley Packet Filter raw network interface

Description The Berkeley Packet Filter provides a raw interface to data link layers in a protocol independent fashion. All packets on the network, even those destined for other hosts, are accessible through this mechanism.

The packet filter appears as a character special device, `/dev/bpf`. After opening the device, the file descriptor must be bound to a specific network interface with the `BIOSETIF` ioctl. A specific interface can be shared by multiple listeners, and the filter underlying each descriptor sees an identical packet stream.

Associated with each open instance of a `bpf` file is a user-settable packet filter. Whenever a packet is received by an interface, all file descriptors listening on that interface apply their filter. Each descriptor that accepts the packet receives its own copy.

Reads from these files return the next group of packets that have matched the filter. To improve performance, the buffer passed to read must be the same size as the buffers used internally by `bpf`. This size is returned by the `BIOCGBLEN` ioctl , and under `BSD`, can be set with `BIOCSBLEN`. An individual packet larger than this size is necessarily truncated.

The packet filter supports any link level protocol that has fixed length headers. Currently, only Ethernet, SLIP and PPP drivers have been modified to interact with `bpf`.

Since packet data is in network byte order, applications should use the `byteorder(3SOCKET)` macros to extract multi-byte values.

A packet can be sent out on the network by writing to a `bpf` file descriptor. The writes are unbuffered, meaning that only one packet can be processed per write. Currently, only writes to Ethernets and SLIP links are supported.

ioctls The `ioctl(2)` command codes in this section are defined in `<net/bfp.h>`. All commands require these includes:

```
#include <sys/types.h>
#include <sys/time.h>
#include <sys/time.h>
#include <net/bpf.h>
```

Additionally, `BIOCGETIF` and `BIOCSETIF` require `<net/if.h>`.

The third argument to the `ioctl(2)`should be a pointer to the type indicated.

`BIOCGBLEN (u_int)`
Returns the required buffer length for reads on `bpf` files.

`BIOCSBLEN (u_int)`
Sets the buffer length for reads on `bpf` files. The buffer must be set before the file is attached to an interface with `BIOCSETIF`. If the requested buffer size cannot be accommodated, the closest allowable size is set and returned in the argument. A read call results in `EINVAL` if it is passed a buffer that is not this size.

`BIOCGDLT (u_int)`
Returns the type of the data link layer underlying the attached interface. `EINVAL` is returned if no interface has been specified. The device types, prefixed with `DLT_`, are defined in `<net/bpf.h>`.

`BIOCGDLTLIST (struct bpf_dltlist)`
Returns an array of available type of the data link layer underlying the attached interface:

```
struct bpf_dltlist {
  u_int bfl_len;
  u_int *bfl_list;
};
```

The available type is returned to the array pointed to the `bfl_list` field while its length in `u_int` is supplied to the `bfl_len` field. `NOMEM` is returned if there is not enough buffer. The `bfl_len` field is modified on return to indicate the actual length in `u_int` of the array returned. If `bfl_list` is `NULL`, the `bfl_len` field is returned to indicate the required length of an array in `u_int`.

`BIOCSDLT (u_int)`
Change the type of the data link layer underlying the attached interface. `EINVAL` is returned if no interface has been specified or the specified type is not available for the interface.

`BIOCPROMISC`
Forces the interface into promiscuous mode. All packets, not just those destined for the local host, are processed. Since more than one file can be listening on a given interface, a listener that opened its interface non-promiscuously can receive packets promiscuously. This problem can be remedied with an appropriate filter.

The interface remains in promiscuous mode until all files listening promiscuously are closed.

`BIOCFLUSH`
Flushes the buffer of incoming packets, and resets the statistics that are returned by `BIOCGSTATS`.

`BIOCGETIF (struct ifreq)`
Returns the name of the hardware interface that the file is listening on. The name is returned in the `ifr_name` field of `ifr`. All other fields are undefined.

`BIOCSETIF (struct ifreq)`
Sets the hardware interface associate with the file. This command must be performed before any packets can be read. The device is indicated by name using the `ifr_name` field of the `ifreq`. Additionally, performs the actions of `BIOCFLUSH`.

`BIOCSRTIMEOUT, BIOCGRTIMEOUT (struct timeval)`
Set or get the read timeout parameter. The `timeval` specifies the length of time to wait before timing out on a read request. This parameter is initialized to zero by open(2), indicating no timeout.

`BIOCGSTATS (struct bpf_stat)`
Returns the following structure of packet statistics:

```
struct bpf_stat {
    uint64_t bs_recv;
    uint64_t bs_drop;
    uint64_t bs_capt;
    uint64_t bs_padding[13];
};
```

The fields are:

`bs_recv` Number of packets received by the descriptor since opened or reset (including any buffered since the last read call.

`bs_drop` Number of packets which were accepted by the filter but dropped by the kernel because of buffer overflows, that is, the application's reads aren't keeping up with the packet traffic.

`bs_capt` Number of packets accepted by the filter.

`BIOCIMMEDIATE (u_int)`
Enable or disable `immediate mode`, based on the truth value of the argument. When immediate mode is enabled, reads return immediately upon packet reception. Otherwise, a read blocks until either the kernel buffer becomes full or a timeout occurs. This is useful for programs like `rarpd(1M)`, which must respond to messages in real time. The default for a new file is `off`.

`BIOCSETF (struct bpf_program)`
Sets the filter program used by the kernel to discard uninteresting packets. An array of instructions and its length is passed in using the following structure:

```
struct bpf_program {
    u_int bf_len;
    struct bpf_insn *bf_insns;
};
```

The filter program is pointed to by the `bf_insns` field while its length in units of `struct bpf_insn` is given by the `bf_len` field. The actions of `BIOCFLUSH` are also performed.

See the `FILTER MACHINE` section of this manual page for an explanation of the filter language.

`BIOCVERSION (struct bpf_version)`
Returns the major and minor version numbers of the filter language currently recognized by the kernel. Before installing a filter, applications must check that the current version is compatible with the running kernel. Version numbers are compatible if the major numbers match and the application minor is less than or equal to the kernel minor. The kernel version number is returned in the following structure:

```
struct bpf_version {
   u_short bv_major;
   u_short bv_minor;
 };
```

The current version numbers are given by `BPF_MAJOR_VERSION` and `BPF_MINOR_VERSION` from `<net/bpf.h>`.

An incompatible filter can result in undefined behavior, most likely, an error returned by `ioctl(2)` or haphazard packet matching.

`BIOCGHDRCMPLT BIOCSHDRCMPLT (u_int)`
Enable/disable or get the `header complete` flag status. If enabled, packets written to the `bpf` file descriptor does not have network layer headers rewritten in the interface output routine. By default, the flag is disabled (value is `0`).

`BIOCGSEESENT BIOCSSEESENT (u_int)`
Enable/disable or get the `see sent` flag status. If enabled, packets sent is passed to the filter. By default, the flag is enabled (value is `1`).

Standard Ioctls `bpf` supports several standard `ioctl(2)`'s that allow the user to do async or non-blocking I/O to an open file descriptor.

`FIONREAD (int)`	Returns the number of bytes that are immediately available for reading.
`SIOCGIFADDR (struct ifreq)`	Returns the address associated with the interface.
`FIONBIO (int)`	Set or clear non-blocking I/O. If `arg` is `non-zero`, then doing a `read(2)` when no data is available returns `-1` and `errno` is set to `EAGAIN`. If `arg` is `zero`, non-blocking I/O is disabled. Setting this overrides the timeout set by `BIOCSRTIMEOUT`.
`FIOASYNC (int)`	Enable or disable async I/O. When enabled (`arg` is `non-zero`), the process or process group specified by `FIOSETOWN` starts receiving `SIGIO`s when packets arrive. You must do an `FIOSETOWN` for this to take effect, as the system does not default this for you. The signal can be changed using `BIOCSRSIG`.
`FIOSETOWN FIOGETOWN (int)`	Set or get the process or process group (if negative) that should receive `SIGIO` when packets are available. The signal can be changed using `BIOCSRSIG`.

`bpf` Header The following structure is prepended to each packet returned by `read(2)`:

```
struct bpf_hdr {
    struct timeval bh_tstamp;
     uint32_t bh_caplen;
```

```
    uint32_t bh_datalen;
    uint16_t bh_hdrlen;
};
```

The fields, whose values are stored in host order, and are:

`bh_tstamp` The time at which the packet was processed by the packet filter.

`bh_caplen` The length of the captured portion of the packet. This is the minimum of the truncation amount specified by the filter and the length of the packet.

`bh_datalen` The length of the packet off the wire. This value is independent of the truncation amount specified by the filter.

`bh_hdrlen` The length of the `BPF` header, which cannot be equal to `sizeof` (`struct bpf_hdr`).

The `bh_hdrlen` field exists to account for padding between the header and the link level protocol. The purpose here is to guarantee proper alignment of the packet data structures, which is required on alignment sensitive architectures and improves performance on many other architectures. The packet filter ensures that the `bpf_hdr` and the network layer header is word aligned. Suitable precautions must be taken when accessing the link layer protocol fields on alignment restricted machines. This is not a problem on an Ethernet, since the `type` field is a short falling on an even offset, and the addresses are probably accessed in a `bytewise` fashion).

Additionally, individual packets are padded so that each starts on a word boundary. This requires that an application has some knowledge of how to get from packet to packet. The macro `BPF_WORDALIGN` is defined in `<net/bpf.h>` to facilitate this process. It rounds up its argument to the nearest `word` aligned value, where a `word` is `BPF_ALIGNMENT` bytes wide.

For example, if `p` points to the start of a packet, this expression advances it to the next packet:

```
p = (char *)p + BPF_WORDALIGN(p->bh_hdrlen + p->bh_caplen)
```

For the alignment mechanisms to work properly, the buffer passed to `read(2)` must itself be word aligned. `malloc(3C)` always returns an aligned buffer.

Filter Machine

A filter program is an array of instructions, with all branches forwardly directed, terminated by a return instruction. Each instruction performs some action on the pseudo-machine state, which consists of an accumulator, index register, scratch memory store, and implicit program counter.

The following structure defines the instruction format:

```
struct bpf_insn {
   uint16_t code;
   u_char  jt;
   u_char  jf;
```

```
   int32_t k;
};
```

The k field is used in different ways by different instructions, and the jt and jf fields are used as offsets by the branch instructions. The opcodes are encoded in a semi-hierarchical fashion. There are eight classes of instructions: BPF_LD, BPF_LDX, BPF_ST, BPF_STX, BPF_ALU, BPF_JMP, BPF_RET, and BPF_MISC. Various other mode and operator bits are or'd into the class to give the actual instructions. The classes and modes are defined in <net/bpf.h>.

Below are the semantics for each defined BPF instruction. We use the convention that A is the accumulator, X is the index register, P[] packet data, and M[] scratch memory store. P[i:n] gives the data at byte offset i in the packet, interpreted as a word (n=4), unsigned halfword (n=2), or unsigned byte (n=1). M[i] gives the i'th word in the scratch memory store, which is only addressed in word units. The memory store is indexed from 0 to BPF_MEMWORDS-1. k, jt, and jf are the corresponding fields in the instruction definition. len refers to the length of the packet.

BPF_LD — These instructions copy a value into the accumulator. The type of the source operand is specified by an addressing mode and can be a constant (BBPF_IMM), packet data at a fixed offset (BPF_ABS), packet data at a variable offset (BPF_IND), the packet length (BPF_LEN), or a word in the scratch memory store (BPF_MEM). For BPF_IND and BPF_ABS, the data size must be specified as a word (BPF_W), halfword (BPF_H), or byte (BPF_B). The semantics of all the recognized BPF_LD instructions follow.

```
BPF_LD+BPF_W+BPF_ABS A <- P[k:4]
BPF_LD+BPF_H+BPF_ABS A <- P[k:2]
BPF_LD+BPF_B+BPF_ABS A <- P[k:1]
BPF_LD+BPF_W+BPF_IND A <- P[X+k:4]
BPF_LD+BPF_H+BPF_IND A <- P[X+k:2]
BPF_LD+BPF_B+BPF_IND A <- P[X+k:1]
BPF_LD+BPF_W+BPF_LEN A <- len
BPF_LD+BPF_IMM A <- k
BPF_LD+BPF_MEM A <- M[k]
```

BPF_LDX — These instructions load a value into the index register. The addressing modes are more restricted than those of the accumulator loads, but they include BPF_MSH, a hack for efficiently loading the IP header length.

```
BPF_LDX+BPF_W+BPF_IMM X <- k
BPF_LDX+BPF_W+BPF_MEM X <- M[k]
BPF_LDX+BPF_W+BPF_LEN X <- len
BPF_LDX+BPF_B+BPF_MSH X <- 4*(P[k:1]&0xf)
```

BPF_ST — This instruction stores the accumulator into the scratch memory. We do not need an addressing mode since there is only one possibility for the destination.

```
BPF_ST M[k] <- A
```

BPF_ALU The alu instructions perform operations between the accumulator and index register or constant, and store the result back in the accumulator. For binary operations, a source mode is required (BPF_K or BPF_X).

```
BPF_ALU+BPF_ADD+BPF_K A <- A + k
BPF_ALU+BPF_SUB+BPF_K A <- A - k
BPF_ALU+BPF_MUL+BPF_K A <- A * k
BPF_ALU+BPF_DIV+BPF_K A <- A / k
BPF_ALU+BPF_AND+BPF_K A <- A & k
BPF_ALU+BPF_OR+BPF_K A <- A | k
BPF_ALU+BPF_LSH+BPF_K A <- A << k
BPF_ALU+BPF_RSH+BPF_K A <- A >> k
BPF_ALU+BPF_ADD+BPF_X A <- A + X
BPF_ALU+BPF_SUB+BPF_X A <- A - X
BPF_ALU+BPF_MUL+BPF_X A <- A * X
BPF_ALU+BPF_DIV+BPF_X A <- A / X
BPF_ALU+BPF_AND+BPF_X A <- A & X
BPF_ALU+BPF_OR+BPF_X A <- A | X
BPF_ALU+BPF_LSH+BPF_X A <- A << X
BPF_ALU+BPF_RSH+BPF_X A <- A >> X
BPF_ALU+BPF_NEG A <- -A
```

BPF_JMP The jump instructions alter flow of control. Conditional jumps compare the accumulator against a constant (BPF_K) or the index register (BPF_X). If the result is true (or non-zero), the true branch is taken, otherwise the false branch is taken. Jump offsets are encoded in 8 bits so the longest jump is 256 instructions. However, the jump always (BPF_JA) opcode uses the 32 bit k field as the offset, allowing arbitrarily distant destinations. All condition also use unsigned comparison conventions.

```
BPF_JMP+BPF_JA  pc += k
BPF_JMP+BPF_JGT+BPF_K  pc += (A > k) ? jt : jf
BPF_JMP+BPF_JGE+BPF_K  pc += (A >= k) ? jt : jf
BPF_JMP+BPF_JEQ+BPF_K  pc += (A == k) ? jt : jf
BPF_JMP+BPF_JSET+BPF_K  pc += (A & k) ? jt : jf
BPF_JMP+BPF_JGT+BPF_X  pc += (A > X) ? jt : jf
BPF_JMP+BPF_JGE+BPF_X  pc += (A >= X) ? jt : jf
BPF_JMP+BPF_JEQ+BPF_X  pc += (A == X) ? jt : jf
BPF_JMP+BPF_JSET+BPF_X  pc += (A & X) ? jt : jf
```

BPF_RET The return instructions terminate the filter program and specify the amount of packet to accept, that is, they return the truncation amount. A return value of zero indicates that the packet should be ignored. The return value is either a constant (BPF_K) or the accumulator (BPF_A).

```
BPF_RET+BPF_A accept A bytes
BPF_RET+BPF_K accept k bytes
```

`BPF_MISC` The miscellaneous category was created for anything that does not fit into the other classes in this section, and for any new instructions that might need to be added. Currently, these are the register transfer instructions that copy the index register to the accumulator or vice versa.

```
BPF_MISC+BPF_TAX X <- A
BPF_MISC+BPF_TXA A <- X
```

The `BPF` interface provides the following macros to facilitate array initializers:

```
BPF_STMT (opcode, operand)
BPF_JUMP (opcode, operand, true_offset, false_offset)
```

Sysctls The following sysctls are available when `bpf` is enabled:

`net.bpf.maxbufsize` Sets the maximum buffer size available for `bpf` peers.

`net.bpf.stats` Shows `bpf` statistics. They can be retrieved with the `netstat(1M)` utility.

`net.bpf.peers` Shows the current `bpf` peers. This is only available to the super user and can also be retrieved with the `netstat(1M)` utility.

Files `/dev/bpf`

Examples **EXAMPLE 1** Using `bfp` to Accept Only Reverse `ARP` Requests

The following example shows a filter taken from the Reverse `ARP` Daemon. It accepts only Reverse `ARP` requests.

```
struct bpf_insn insns[] = {
           BPF_STMT(BPF_LD+BPF_H+BPF_ABS, 12),
           BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, ETHERTYPE_REVARP, 0, 3),
           BPF_STMT(BPF_LD+BPF_H+BPF_ABS, 20),
           BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, REVARP_REQUEST, 0, 1),
           BPF_STMT(BPF_RET+BPF_K, sizeof(struct ether_arp) +
               sizeof(struct ether_header)),
           BPF_STMT(BPF_RET+BPF_K, 0),
};
```

EXAMPLE 2 Using `bfp` to Accept IP Packets

The following example shows filter that accepts only IP packets between host `128.3.112.15` and `128.3.112.35`.

```
struct bpf_insn insns[] = {
           BPF_STMT(BPF_LD+BPF_H+BPF_ABS, 12),
           BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, ETHERTYPE_IP, 0, 8),
           BPF_STMT(BPF_LD+BPF_W+BPF_ABS, 26),
           BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, 0x8003700f, 0, 2),
```

EXAMPLE 2 Using `bfp` to Accept IP Packets *(Continued)*

```
            BPF_STMT(BPF_LD+BPF_W+BPF_ABS, 30),
            BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, 0x80037023, 3, 4),
            BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, 0x80037023, 0, 3),
            BPF_STMT(BPF_LD+BPF_W+BPF_ABS, 30),
            BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, 0x8003700f, 0, 1),
            BPF_STMT(BPF_RET+BPF_K, (u_int)-1),
            BPF_STMT(BPF_RET+BPF_K, 0),
};
```

EXAMPLE 3 Using `bfp` to Return Only TCP Finger Packets

The following example shows a filter that returns only TCP finger packets. The IP header must be parsed to reach the TCP header. The `BPF_JSET` instruction checks that the `IP` fragment offset is 0 so we are sure that we have a `TCP` header.

```
struct bpf_insn insns[] = {
            BPF_STMT(BPF_LD+BPF_H+BPF_ABS, 12),
            BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, ETHERTYPE_IP, 0, 10),
            BPF_STMT(BPF_LD+BPF_B+BPF_ABS, 23),
            BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, IPPROTO_TCP, 0, 8),
            BPF_STMT(BPF_LD+BPF_H+BPF_ABS, 20),
            BPF_JUMP(BPF_JMP+BPF_JSET+BPF_K, 0x1fff, 6, 0),
            BPF_STMT(BPF_LDX+BPF_B+BPF_MSH, 14),
            BPF_STMT(BPF_LD+BPF_H+BPF_IND, 14),
            BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, 79, 2, 0),
            BPF_STMT(BPF_LD+BPF_H+BPF_IND, 16),
            BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, 79, 0, 1),
            BPF_STMT(BPF_RET+BPF_K, (u_int)-1),
            BPF_STMT(BPF_RET+BPF_K, 0),
};
```

Attributes

See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	Sparc, x86
Interface Stability	Committed

See Also

`netstat(1M)`, `rarpd(1M)`, `lseek(2)`, `ioctl(2)`, `open(2)`, `read(2)`, `malloc(3C)`, `select(3C)`, `byteorder(3SOCKET)`, `signal(3C)`, `attributes(5)`

S. McCanne and V. Jacobson, *The BSD Packet Filter: A New Architecture for User-level Packet Capture*, Proceedings of the 1993 Winter USENIX.

Bugs The read buffer must be of a fixed size returned by the `BIOCGBLEN` ioctl.

A file that does not request promiscuous mode can receive promiscuous received packets as a side effect of another file requesting this mode on the same hardware interface. This could be fixed in the kernel with additional processing overhead. However, we favor the model where all files must assume that the interface is promiscuous, and if so desired, must use a filter to reject foreign packets.

Data link protocols with variable length headers are not currently supported.

Under SunOS, if a `BPF` application reads more than `2^31` bytes of data, read fails in `EINVAL`signal(3C). You can either fix the bug in SunOS, or `lseek(2)` to `0` when read fails for this reason.

`Immediate mode` and the `read timeout` are misguided features. This functionality can be emulated with non-blocking mode and `select(3C)`.

Name bpp – bi-directional parallel port driver

Synopsis SUNW,bpp@*slot*,*offset*:bpp*n*

Description The bpp driver provides a general-purpose bi-directional interface to parallel devices. It supports a variety of output (printer) and input (scanner) devices, using programmable timing relationships between the various handshake signals.

Application Programming Interface The bpp driver is an *exclusive-use* device. If the device has already been opened, subsequent opens fail with EBUSY.

Default Operation Each time the bpp device is opened, the default configuration is BPP_ACK_BUSY_HS for read handshake, BPP_ACK_HS for write handshake, 1 microsecond for all setup times and strobe widths, and 60 seconds for both timeouts. This configuration (in the write mode) drives many common personal computer parallel printers with Centronics-type interfaces. The application should use the BPPIOC_SETPARMS ioctl request to configure the bpp for the particular device which is attached, if necessary.

Write Operation If a failure or error condition occurs during a write(2), the number of bytes successfully written is returned (short write). Note that errno will not be set. The contents of certain status bits will be captured at the time of the error, and can be retrieved by the application program, using the BPPIOC_GETERR ioctl request. Subsequent write(2) calls may fail with the system error ENXIO if the error condition is not rectified. The captured status information will be overwritten each time an attempted transfer or a BPPIOC_TESTIO ioctl request occurs.

Read Operations If a failure or error condition occurs during a read(2), the number of bytes successfully read is returned (short read). Note that errno will not be set. The contents of certain status bits will be captured at the time of the error, and can be retrieved by the application, using the BPPIOC_GETERR ioctl request. Subsequent read(2) calls may fail with ENXIO if the error condition is not rectified. The captured register information will be overwritten each time an attempted transfer or a BPPIOC_TESTIO ioctl request.

If the read_handshake element of the bpp_transfer_parms structure (see below) is set to BPP_CLEAR_MEM or BPP_SET_MEM, zeroes or ones, respectively, are written into the user buffer.

Read/Write Operation When the driver is opened for reading and writing, it is assumed that scanning will take place, as scanners are the only devices supported by this mode. Most scanners require that the SLCT_IN or AFX pin be set to tell the scanner the direction of the transfer. The AFX line is set when the read_handshake element of the bpp_transfer_parms structure is set to BPP_HSCAN_HS, otherwise the SLCT_IN pin is set. Normally, scanning starts by writing a command to the scanner, at which time the pin is set. When the scan data is read back, the pin is reset.

ioctls The following ioctl requests are supported:

BPPIOC_SETPARMS Set transfer parameters.

The argument is a pointer to a `bpp_transfer_parms` structure. See below for a description of the elements of this structure. If a parameter is out of range, `EINVAL` is returned.

`BPPIOC_GETPARMS` Get current transfer parameters.

The argument is a pointer to a `bpp_transfer_parms` structure. See below for a description of the elements of this structure. If no parameters have been configured since the device was opened, the contents of the structure will be the default conditions of the parameters (see `Default Operation` above).

`BPPIOC_SETOUTPINS` Set output pin values.

The argument is a pointer to a `bpp_pins` structure. See below for a description of the elements of this structure. If a parameter is out of range, `EINVAL` is returned.

`BPPIOC_GETOUTPINS` Read output pin values. The argument is a pointer to a `bpp_pins` structure. See below for a description of the elements of this structure.

`BPPIOC_GETERR` Get last error status.

The argument is a pointer to a `bpp_error_status` structure. See below for a description of the elements of this structure. This structure indicates the status of all the appropriate status bits at the time of the most recent error condition during a `read(2)` or `write(2)` call, or the status of the bits at the most recent `BPPIOC_TESTIO` ioctl request. Note: The bits in the `pin_status` element indicate whether the associated pin is active, not the actual polarity. The application can check transfer readiness without attempting another transfer using the `BPPIOC_TESTIO` ioctl. Note: The `timeout_occurred` and `bus_error` fields will never be set by the `BPPIOC_TESTIO` ioctl, only by an actual failed transfer.

`BPPIOC_TESTIO` Test transfer readiness.

This command checks to see if a read or write transfer would succeed based on pin status, opened mode, and handshake selected. If a handshake would succeed, `0` is returned. If a transfer would fail, `-1` is returned, and `errno` is set to `EIO,` and the error status information is captured. The captured status can be retrieved using the `BPPIOC_GETERR` ioctl call. Note that the `timeout_occurred` and `bus_error` fields will never be set by this ioctl.

Transfer Parameters Structure

This structure is defined in <sys/bpp_io.h>.

```
struct bpp_transfer_parms {
    enum  handshake_t
         read_handshake;      /* parallel port read handshake mode */
    int  read_setup_time;    /* DSS register - in nanoseconds */
    int  read_strobe_width;  /* DSW register - in nanoseconds */
    int  read_timeout;       /*
                              * wait this many seconds
                              * before aborting a transfer
                             */
    enum    handshake_t
         write_handshake;      /* parallel port write handshake mode */
    int  write_setup_time;    /* DSS register - in nanoseconds */
    int  write_strobe_width;  /* DSW register - in nanoseconds */
    int  write_timeout;       /*
                               * wait this many seconds
                               * before aborting a transfer
                              */
};
/* Values for read_handshake and write_handshake fields */
    enum    handshake_t {
        BPP_NO_HS,            /* no handshake pins */
        BPP_ACK_HS,           /* handshake controlled by ACK line */
        BPP_BUSY_HS,          /* handshake controlled by BSY line */
        BPP_ACK_BUSY_HS,      /*
                               * handshake controlled by ACK and BSY lines
                               * read_handshake only!
                              */
        BPP_XSCAN_HS,         /* xerox scanner mode,
                               * read_handshake only!
                                 */
        BPP_HSCAN_HS,         /*
                               * HP scanjet scanner mode
                               * read_handshake only!
                              */
        BPP_CLEAR_MEM,        /* write 0's to memory,
                               * read_handshake only!
                              */
        BPP_SET_MEM,          /* write 1's to memory,
                               * read_handshake only!
                              */
    /* The following handshakes are RESERVED. Do not use. */
        BPP_VPRINT_HS,        /* valid only in read/write mode */
        BPP_VPLOT_HS          /* valid only in read/write mode */
};
```

The read_setup_time field controls the time between dstrb falling edge to bsy rising edge if the read_handshake field is set to BPP_NO_HS or BPP_ACK_HS. It controls the time between dstrb falling edge to ack rising edge if the read_handshake field is set to BPP_ACK_HS or BPP_ACK_BUSY_HS. It controls the time between ack falling edge to dstrb rising edge if the read_handshake field is set to BPP_XSCAN_HS.

The read_strobe_width field controls the time between ack rising edge and ack falling edge if the read_handshake field is set to BPP_NO_HS or BPP_ACK_BUSY_HS. It controls the time between dstrb rising edge to dstrb falling edge if the read_handshake field is set to BPP_XSCAN_HS.

The values allowed for the write_handshake field are duplicates of the definitions for the read_handshake field. Note that some of these handshake definitions are only valid in one mode or the other.

The write_setup_time field controls the time between data valid to dstrb rising edge for all values of the write_handshake field.

The write_strobe_width field controls the time between dstrb rising edge and dstrb falling edge if the write_handshake field is not set to BPP_VPRINT_HS or BPP_VPLOT_HS. It controls the minimum time between dstrb rising edge to dstrb falling edge if the write_handshake field is set to BPP_VPRINT_HS or BPP_VPLOT_HS.

Transfer Pins Structure

This structure is defined in <sys/bpp_io.h>.

```
struct  bpp_pins {
    uchar_t  output_reg_pins;  /* pins in P_OR register */
    uchar_t  input_reg_pins;   /* pins in P_IR register */
};

/* Values for output_reg_pins field */
#define BPP_SLCTIN_PIN  0x01  /* Select in pin */
#define BPP_AFX_PIN     0x02  /* Auto feed pin */
#define BPP_INIT_PIN    0x04  /* Initialize pin */
#define BPP_V1_PIN      0x08  /* reserved pin 1 */
#define BPP_V2_PI       0x10  /* reserved pin 2 */
#define BPP_V3_PIN      0x20  /* reserved pin 3 */
#define BPP_ERR_PIN     0x01  /* Error pin */
#define BPP_SLCT_PIN    0x02  /* Select pin */
#define BPP_PE_PIN      0x04  /* Paper empty pin */
```

Error Pins Structure

This structure is defined in the include file <sys/bpp_io.h>.

```
struct bpp_error_status {
    char    timeout_occurred;    /* 1 if a timeout occurred */
    char    bus_error;    /* 1 if an SBus bus error */
    uchar_t pin_status;    /*
             * status of pins which could
```

```
                * cause an error
                */
};
/* Values for pin_status field */
#define BPP_ERR_ERR    0x01    /* Error pin active */
#define BPP_SLCT_ERR   0x02    /* Select pin active */
#define BPP_PE_ERR     0x04    /* Paper empty pin active */
#define BPP_SLCTIN_ERR 0x10    /* Select in pin active    */
#define BPP_BUSY_ERR   0x40    /* Busy pin active */
```

Errors

`EBADF` The device is opened for write-only access and a read is attempted, or the device is opened for read-only access and a write is attempted.

`EBUSY` The device has been opened and another open is attempted. An attempt has been made to unload the driver while one of the units is open.

`EINVAL` A `BPPIOC_SETPARMS` `ioctl` is attempted with an out of range value in the `bpp_transfer_parms` structure. A `BPPIOC_SETOUTPINS` `ioctl` is attempted with an invalid value in the pins structure. An `ioctl` is attempted with an invalid value in the command argument. An invalid command argument is received during `modload(1M)` or `modunload(1M)`.

`EIO` The driver encountered an SBus bus error when attempting an access.

A read or write does not complete properly, due to a peripheral error or a transfer timeout.

A `BPPIOC_TESTIO` `ioctl` call is attempted while a condition exists which would prevent a transfer (such as a peripheral error).

`ENXIO` The driver has received an open request for a unit for which the attach failed. The driver has received a read or write request for a unit number greater than the number of units available. The driver has received a write request for a unit which has an active peripheral error.

Files

`/dev/bpp`*n* bi-directional parallel port devices

See Also

`ioctl(2)`, `read(2)`, `write(2)`, `sbus(4)`

Name bscv, bscbus, i2bsc – Blade support chip interface driver

Description The `bscv`, `bscbus` and `i2bsc` drivers interface with the Blade support chip used on Sun Microsystem's Blade server products. These drivers provide a conduit for passing control, environmental, cpu signature and event information between Solaris and the Blade support chip.

These drivers do not export public interfaces. Instead they make information available via picl, prtdiag, prtfru and related tools. In addition, these drivers log Blade support chip environmental event information into system logs.

Files

`/platform/sun4u/kernel/drv/sparcv9/bscbus`	64-bit ELF kernel driver
`/platform/sun4u/kernel/drv/sparcv9/bscv`	64-bit ELF kernel driver
`/platform/sun4u/kernel/drv/sparcv9/i2bsc`	64-bit ELF kernel driver
`/platform/i86pc/kernel/drv/bscbus`	32–bit ELF kernel file (x86 only)
`/platform/i86pc/kernel/drv/bscv`	32–bit ELF kernel file (x86 only)

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	Limited to systems with Blade Support Chip
Availability	SUNWcar.u, SUNWcar.i

Name bufmod – STREAMS Buffer Module

Synopsis

```
#include <sys/bufmod.h>

ioctl(fd, I_PUSH, "bufmod");
```

Description `bufmod` is a STREAMS module that buffers incoming messages, reducing the number of system calls and the associated overhead required to read and process them. Although `bufmod` was originally designed to be used in conjunction with STREAMS-based networking device drivers, the version described here is general purpose so that it can be used anywhere STREAMS input buffering is required.

Read-side Behavior The behavior of `bufmod` depends on various parameters and flags that can be set and queried as described below under `IOCTLS`. `bufmod` collects incoming `M_DATA` messages into chunks, passing each chunk upstream when the chunk becomes full or the current read timeout expires. It optionally converts `M_PROTO` messages to `M_DATA` and adds them to chunks as well. It also optionally adds to each message a header containing a timestamp, and a cumulative count of messages dropped on the stream read side due to resource exhaustion or flow control. Thedefault settings of `bufmod` allow it to drop messages when flow control sets in or resources are exhausted; disabling headers and explicitly requesting no drops makes `bufmod` pass all messages through. Finally, `bufmod` is capable of truncating upstream messages to a fixed, programmable length.

When a message arrives, `bufmod` processes it in several steps. The following paragraphs discuss each step in turn.

Upon receiving a message from below, if the `SB_NO_HEADER` flag is not set, `bufmod` immediately timestamps it and saves the current time value for later insertion in the header described below.

Next, if `SB_NO_PROTO_CVT` is not set, `bufmod` converts all leading `M_PROTO` blocks in the message to `M_DATA` blocks, altering only the message type field and leaving the contents alone.

It then truncates the message to the current *snapshot length*, which is set with the `SBIOCSSNAP` `ioctl` described below.

Afterwards, if `SB_NO_HEADER` is not set, `bufmod` prepends a header to the converted message. This header is defined as follows.

```
struct sb_hdr {
        uint_t   sbh_origlen;
        uint_t   sbh_msglen;
        uint_t   sbh_totlen;
        uint_t   sbh_drops;
#if defined(_LP64) || defined(_I32LPx)
        struct  timeval32 sbh_timestamp;
#else
        struct  timeval sbh_timestamp;
#endif /* !_LP64 */
};
```

The sbh_origlen field gives the message's original length before truncation in bytes. The sbh_msglen field gives the length in bytes of the message after the truncation has been done. sbh_totlen gives the distance in bytes from the start of the truncated message in the current chunk (described below) to the start of the next message in the chunk; the value reflects any padding necessary to insure correct data alignment for the host machine and includes the length of the header itself. sbh_drops reports the cumulative number of input messages that this instance of bufmod has dropped due to flow control or resource exhaustion. In the current implementation message dropping due to flow control can occur only if the SB_NO_DROPS flag is not set. (Note: this accounts only for events occurring within bufmod, and does not count messages dropped by downstream or by upstream modules.) The sbh_timestamp field contains the message arrival time expressed as a struct timeval.

After preparing a message, bufmod attempts to add it to the end of the current chunk, using the chunk size and timeout values to govern the addition. The chunk size and timeout values are set and inspected using the ioctl() calls described below. If adding the new message would make the current chunk grow larger than the chunk size, bufmod closes off the current chunk, passing it up to the next module in line, and starts a new chunk. If adding the message would still make the new chunk overflow, the module passes it upward in an over-size chunk of its own. Otherwise, the module concatenates the message to the end of the current chunk.

To ensure that messages do not languish forever in an accumulating chunk, bufmod maintains a read timeout. Whenever this timeout expires, the module closes off the current chunk and passes it upward. The module restarts the timeout period when it receives a read side data message and a timeout is not currently active. These two rules insure that bufmod minimizes the number of chunks it produces during periods of intense message activity and that it periodically disposes of all messages during slack intervals, but avoids any timeout overhead when there is no activity.

bufmod handles other message types as follows. Upon receiving an M_FLUSH message specifying that the read queue be flushed, the module clears the currently accumulating chunk and passes the message on to the module or driver above. (Note: bufmod uses zero length M_CTL messages for internal synchronization and does not pass them through.) bufmod passes all other messages through unaltered to its upper neighbor, maintaining message order for non high priority messages by passing up any accumulated chunk first.

If the SB_DEFER_CHUNK flag is set, buffering does not begin until the second message is received within the timeout window.

If the SB_SEND_ON_WRITE flag is set, bufmod passes up the read side any buffered data when a message is received on the write side. SB_SEND_ON_WRITE and SB_DEFER_CHUNK are often used together.

Write-side Behavior

bufmod intercepts M_IOCTL messages for the ioctls described below. The module passes all other messages through unaltered to its lower neighbor. If SB_SEND_ON_WRITE is set, message arrival on the writer side suffices to close and transmit the current read side chunk.

ioctls

bufmod responds to the following ioctls.

`SBIOCSTIME` Set the read timeout value to the value referred to by the `struct timeval` pointer given as argument. Setting the timeout value to zero has the side-effect of forcing the chunk size to zero as well, so that the module will pass all incoming messages upward immediately upon arrival. Negative values are rejected with an `EINVAL` error.

`SBIOCGTIME` Return the read timeout in the `struct timeval` pointed to by the argument. If the timeout has been cleared with the `SBIOCCTIME` `ioctl`, return with an `ERANGE` error.

`SBIOCCTIME` Clear the read timeout, effectively setting its value to infinity. This results in no timeouts being active and the chunk being delivered when it is full.

`SBIOCSCHUNK` Set the chunk size to the value referred to by the *uint_t* pointer given as argument. See Notes for a description of effect on stream head high water mark.

`SBIOCGCHUNK` Return the chunk size in the *uint_t* pointed to by the argument.

`SBIOCSSNAP` Set the current snapshot length to the value given in the `uint_t` pointed to by the `ioctl`'s final argument. `bufmod` interprets a snapshot length value of zero as meaning infinity, so it will not alter the message. See Notes for a description of effect on stream head high water mark.

`SBIOCGSNAP` Returns the current snapshot length in the `uint_t` pointed to by the `ioctl`'s final argument.

`SBIOCSFLAGS` Set the current flags to the value given in the `uint_t` pointed to by the `ioctl`'s final argument. Possible values are a combination of the following.

- `SB_SEND_ON_WRITE` Transmit the read side chunk on arrival of a message on the write side.
- `SB_NO_HEADER` Do not add headers to read side messages.
- `SB_NO_DROPS` Do not drop messages due to flow control upstream.
- `SB_NO_PROTO_CVT` Do not convert `M_PROTO` messages into `M_DATA`.
- `SB_DEFER_CHUNK` Begin buffering on arrival of the second read side message in a timeout interval.

`SBIOCGFLAGS` Returns the current flags in the `uint_t` pointed to by the `ioctl`'s final argument.

See Also `dlpi(7P)`, `pfmod(7M)`

Notes Older versions of `bufmod` did not support the behavioral flexibility controlled by the `SBIOCSFLAGS` `ioctl`. Applications that wish to take advantage of this flexibility can guard themselves against old versions of the module by invoking the `SBIOCGFLAGS` ioctl and checking for an `EINVAL` error return.

When buffering is enabled by issuing an `SBIOCSCHUNK` ioctl to set the chunk size to a non zero value, `bufmod` sends a `SETOPTS` message to adjust the stream head high and low water marks to accommodate the chunked messages.

When buffering is disabled by setting the chunk size to zero, message truncation can have a significant influence on data traffic at the stream head and therefore the stream head high and low water marks are adjusted to new values appropriate for the smaller truncated message sizes.

Bugs `bufmod` does not defend itself against allocation failures, so that it is possible, although very unlikely, for the stream head to use inappropriate high and low water marks after the chunk size or snapshot length have changed.

Name cadp160 – Adaptec Ultra160 SCSI host bus adapter driver

Synopsis `scsi@unit-address`

Description The `cadp160` host bus adapter driver is a SCSA-compliant nexus driver that supports the following Adaptec Ultra160 SCSI devices:

- Adapters: 39160, 29160, 29160N, 29160LP
- Chips: AIC-7892B1, AIC-7899A, AIC-7899B2

The `cadp160` driver supports standard functions provided by the SCSA interface including tagged and untagged queuing, wide, fast and ultra SCSI, and auto request sense. The `cadp160` driver does not support linked commands. The `cadp160` driver supports hot swap SCSI, hot plug PCI, 64-bit addressing (dual address cycle), domain validation, PCI bus clock rates up to 66MHZ and narrow and wide devices at 20MB/sec, 40MB/sec, 80MB/sec, and 160MB/sec.

Files

`/platform/i86pc/kernel/drv/cadp160`	32–bit ELF kernel module.
`/platform/i86pc/kernel/drv/amd64/cadp160`	64–bit ELF kernel module.
`/platform/i86pc/kernel/drv/cadp160.conf`	Optional configuration file.

Attributes See `attributes(5)` for a description of the following attribute:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86

See Also `prtconf(1M)`, `driver.conf(4)`, `pci(4)`, `attributes(5)`, `dlpi(7P)`, `scsi_abort(9F)`, `scsi_hba_attach_setup(9F)`, `scsi_ifgetcap(9F)`, `scsi_ifsetcap(9F)`, `scsi_reset(9F)`, `scsi_sync_pkt(9F)`, `scsi_transport(9F)`, `scsi_device(9S)`, `scsi_extended_sense(9S)`, `scsi_inquiry(9S)`, `scsi_pkt(9S)`

Writing Device Drivers

Solaris (x86 Edition) Hardware Compatibility List

ANSI Small Computer System Interface-2 (SCSI-2)

Name cadp – Adaptec Ultra-2 SCSI host bus adapter driver

Synopsis `scsi@unit-address`

Description The `cadp` host bus adapter driver is a SCSA–compliant nexus driver that supports the following Adaptec Ultra-2 SCSI devices:

- Adapters: Adaptec AHA-2940U2W, AHA-2940U2B, AHA-2940U2, AHA-2950U2B, AHA-3950U2B
- Chips: AIC-7896/AIC-7897, AIC-7890/AIC-7890A, AIC-7891, AIC-7890AB, AIC-7890A

The `cadp` driver supports standard functions provided by the SCSA interface, including tagged and untagged queuing, Wide/Fast/Ultra SCSI, and auto request sense. The cadp driver does not support linked commands.

Preconfigure

- The Plug N Play SCAM Support option is not supported.
- If the BIOS is enabled on the card, ensure that the Adaptec SCSISelect BIOS option Reset SCSI Bus at IC Initialization (under the Advanced Configuration Options menu) is set to `Enabled`. Run the SCSISelect utility by pressing `Ctrl-A` when you see the Adaptec banner during system boot.
- If the adapter is being used in a multi-initiator configuration, do the following: (1) Ensure that the system boot disk is not on the shared (clustered) bus. (2) Set the Reset SCSI Bus at IC Initialization option to `Disabled`. (3) Set the Host Adapter BIOS option (under the Advanced Configuration Options menu) to `Disabled:Not scan`. (4) Add the `allow-bus-reset=0` property to the `/kernel/drv/cadp.conf` file.
- Reboot the system after you install patches.

Known Problems and Limitations

- Running the `format(1M)` command on a Seagate ST19171W 9 GB disk drive fails.
- Some motherboards may have problems supporting channel B with boards based on the Adaptec AIC-7896 chip. The problem arises because the BIOS doesn't properly assign two interrupts for PCI interrupts INTA and INTB on the slot containing the AIC-7896 chip. As a result, timeouts and resets on those devices appear on the console. For some motherboards, you can work around the problem by setting the Advanced/PCI IRQ Mapping feature to ISA Legacy IRQs.
- If you experience problems when using a narrow SCSI CD-ROM drive on the internal wide interface, disable "negotiate wide," "negotiate sync," or both for that device in the Adaptec configuration utility.
- The Fujitsu narrow disk (M1603SAU) can reselect with an invalid queue tag ID. This violates the SCSI protocol and it causes the `cadp` driver to behave erroneously. Because this is difficult to guard against, you should disable tagged queuing for these targets. Use the `iostat --E` command to determine if you have a Fujitsu M1603S-512 disk. If you do, edit the `/kernel/drv/cadp.conf` file and add the property `target`*n*`-scsi-options=0x1f78`, where *n* is the target number.
- The IBM external wide disk (DFHSS2W, Revision 1717) is not supported.

- When setting up a SCSI bus configuration, avoid connecting wide devices to a narrow bus. However, if you have such a configuration, add the following entry to the `cadp.conf` file: `target`*n*`-scsi-options=0x1df8` where *n* is the target ID of the wide device on the narrow bus. This entry disables wide negotiation for the specified target. Also ensure that the upper 8 bits of the bus are properly terminated at both ends of the SCSI chain.
- If you experience installation problems on systems with Intel 440BX/440GX motherboards, upgrade the motherboard BIOS with the latest revision.

Configuration You configure the `cadp` host bus adapter driver by defining the properties found in `cadp.conf`. The `cadp.conf` file contains properties that you can modify, including: `scsi-options`, `target<`*n*`>-scsi-options`, `scsi-reset-delay`, and `scsi-initiator-id`. Properties in the `cadp.conf` file override global SCSI settings.

The property `target<`*n*`>-scsi-options` overrides the `scsi-options` property value for `target<`*n*`>`, where <*n*> can vary from decimal *0* to *15*. The `cadp` driver supports the following scsi-options: `SCSI_OPTIONS_DR`, `SCSI_OPTIONS_SYNC`, `SCSI_OPTIONS_TAG`, `SCSI_OPTIONS_FAST`, `SCSI_OPTIONS_WIDE`, `SCSI_OPTIONS_FAST20`, and `SCSI_OPTIONS_FAST40`.

You configure the SCSI devices using the Adaptec configuration utility. When configuring the devices, you should observe the following guidelines:

- Configure each device using a unique SCSI ID. On the Advanced Configuration Options menu, set Plug N Play SCAM Support to `Disabled`. Ensure that devices on either end of the SCSI chain are terminated. When mixing wide (16 bits) and narrow (8 bits) devices on the same wide chain, ensure that a wide device is at the end of the chain. If you place a narrow device at the end of the chain, wide devices on the same chain will terminate the low byte, resulting in a illegal configuration.
- If there is more than one controller, or an embedded controller, attempt to use one IRQ per controller.
- When prompted, enable bus mastering for the slot(s) with your host bus adapter(s.)
- Enable support for disks larger than 1 Gbyte, if applicable.

Examples Create a file called `/kernel/drv/cadp.conf`, then add the following line:

```
scsi-options=0x78;
```

The above line disables tagged queuing, Fast/Ultra SCSI, and wide mode for all `cadp` instances.

To set `scsi-options` more specifically per target, add the following lines to `/kernel/drv/cadp.conf`:

```
target1-scsi-options=0x78;
device-type-scsi-options-list =
"SEAGATE ST32550W", "seagate-scsi-options" ;
```

```
seagate-scsi-options = 0x58;
scsi-options=0x3f8;
```

With the exception of one disk type that has `scsi-options` set to `0x58`, the above example sets `scsi-options` for target 1 to 0x78, and all remaining targets to 0x3f8.

The `scsi-options` properties that are specified per target ID have the highest precedence, followed by `scsi-options` per device type. Global `scsi-options` for all `cadp` instances per bus have the lowest precedence. You must reboot the system for the specified `scsi options` to take effect.

Driver Capabilities

To enable certain features on the `cadp` driver, the target driver must set capabilities. The following capabilities can be queried and modified by the target driver: `synchronous`, `tagged-qing`, `wide-xfer`, `auto-rqsense`, `qfull-retries`, and `qfull-retry-interval`. All other capabilities are query only.

By default, the `tagged-qing`, `auto-rqsense`, and `wide-xfer` capabilities are disabled. The `disconnect`, `synchronous`, and `untagged-qing` capabilities are always enabled. The `cadp` driver capabilities can only be assigned binary values (0 or 1). The default value for `qfull-retries` is *10* and the default value for `qfull-retry-interval` is *100*. The `qfull-retries` capability is `au_char` (0 to 255) while `qfull-retry-interval` is a `u_short` (0 to 65535).

If a conflict occurs between the value of `scsi-options` and a capability, the value set in `scsi-options` prevails. Only `whom != 0` is supported in the `scsi_ifsetcap(9F)` call. See `scsi_ifsetcap(9F)` and `scsi_ifgetcap(9F)` for details.

Files

`/kernel/drv/cadp`	ELF kernel module
`/kernel/drv/cadp.conf`	Optional configuration file

Attributes

See `attributes(5)` for a description of the following attribute:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86

See Also

`prtconf(1M)`, `driver.conf(4)`, `pci(4)`, `attributes(5)`, `scsi_abort(9F)`, `scsi_hba_attach(9F)`, `scsi_ifgetcap(9F)`, `scsi_ifsetcap(9F)`, `scsi_reset(9F)`, `scsi_sync_pkt(9F)`, `scsi_transport(9F)`, `scsi_device(9S)`, `scsi_extended_sense(9S)`, `scsi_inquiry(9S)`, `scsi_pkt(9S)`

Writing Device Drivers

Solaris (Intel Platform Edition) Hardware Compatibility List

ANSI Small Computer System Interface-2 (SCSI-2)

Notes The `cadp` driver supports the adapters and chipsets listed in this man page. For information on support of additional devices, see the *Solaris (Intel Platform Edition) Hardware Compatibility List* a component of the *Information Library for Solaris 8 (Intel Platform Edition).*

The `cadp` driver exports properties indicating (per target) the negotiated transfer speed (`target<n>-sync-speed`), whether wide bus (`target<n>-wide`), is supported for that particular target (`target<n>-scsi-options`), and whether tagged queuing (`target<n>-tag-queue`) has been enabled. The `sync-speed` property value is the data transfer rate in KB/sec. The `target<n>-tag-queue` and the `target<n>-wide` property have value 1 to indicate that the corresponding capability is enabled, or 0 to indicate that the capability is disabled. See `prtconf(1M)` (verbose option) for information on viewing the `cadp` properties.

Sample output is provided below:

```
pci9005,f500, instance #2
System software properties:
     name <interrupt-priorities> length <4>
         value <0x05000000>.
     name <tape> length <5>
         value <0x7363747000>.
     name <disk> length <5>
         value <0x7363646b00>.
     name <queue> length <6>
         value <0x71736f727400>.
     name <flow_control>  length <6>
         value <0x646d756c7400>.
Driver properties:
     name <target0-tag-queue> length <4>
         value <0x01000000>.
     name <target0-wide> length <4>
         value <0x01000000>.
     name <target0-sync-speed> length <4>
         value <0x28000000>.
     name <chosen-interrupt> length <8>
         value <0x0100000000000000>.
     name <scsi-selection-timeout> length <4>
         value <0xfa000000>.
     name <scsi-options> length <4>
         value <0xf81f0000>.
     name <scsi-watchdog-tick> length <4>
         value <0x0a000000>.
     name <scsi-tag-age-limit> length <4>
         value <0x02000000>.
     name <scsi-reset-delay> length <4>
         value <0xb80b0000>.
```

Name cdio – CD-ROM control operations

Synopsis

```
#include <sys/cdio.h>
```

Description The set of ioctl(2) commands described below are used to perform audio and CD-ROM specific operations. Basic to these cdio ioctl requests are the definitions in <sys/cdio.h>.

Several CD-ROM specific commands can report addresses either in LBA (Logical Block Address) format or in MSF (Minute, Second, Frame) format. The READ HEADER, READ SUBCHANNEL, and READ TABLE OF CONTENTS commands have this feature.

LBA format represents the logical block address for the CD-ROM absolute address field or for the offset from the beginning of the current track expressed as a number of logical blocks in a CD-ROM track relative address field. MSF format represents the physical address written on CD-ROM discs, expressed as a sector count relative to either the beginning of the medium or the beginning of the current track.

ioctls The following I/O controls do not have any additional data passed into or received from them.

CDROMSTART — This ioctl() spins up the disc and seeks to the last address requested.

CDROMSTOP — This ioctl() spins down the disc.

CDROMPAUSE — This ioctl() pauses the current audio play operation.

CDROMRESUME — This ioctl() resumes the paused audio play operation.

CDROMEJECT — This ioctl() ejects the caddy with the disc.

CDROMCLOSETRAY — This ioctl() closes the caddy with the disc.

The following I/O controls require a pointer to the structure for that ioctl(), with data being passed into the ioctl().

CDROMPLAYMSF — This ioctl() command requests the drive to output the audio signals at the specified starting address and continue the audio play until the specified ending address is detected. The address is in MSF format. The third argument of this ioctl() call is a pointer to the type struct cdrom_msf.

```
/*
 * definition of play audio msf structure
 */
struct cdrom_msf {
    unsigned char    cdmsf_min0;    /* starting minute*/
    unsigned char    cdmsf_sec0;    /* starting second*/
    unsigned char    cdmsf_frame0;    /*starting frame*/
    unsigned char    cdmsf_min1;    /* ending minute */
    unsigned char    cdmsf_sec1;    /* ending second */
    unsigned char    cdmsf_frame1;    /* ending frame */
};
```

The CDROMREADTOCENTRY ioctl request may be used to obtain the start time for a track. An approximation of the finish time can be obtained by using the CDROMREADTOCENTRY ioctl request to retrieve the start time of the track following the current track.

The leadout track is the next consecutive track after the last audio track. Hence, the start time of the leadout track may be used as the effective finish time of the last audio track.

CDROMPLAYTRKIND — This `ioctl()` command is similar to CDROMPLAYMSF. The starting and ending address is in track/index format. The third argument of the `ioctl()` call is a pointer to the type `struct cdrom_ti`.

```
/*
 * definition of play audio track/index structure
 */
struct cdrom_ti {
    unsigned char    cdti_trk0;    /* starting track*/
    unsigned char    cdti_ind0;    /* starting index*/
    unsigned char    cdti_trk1;    /* ending track */
    unsigned char    cdti_ind1;    /* ending index */
};
```

CDROMVOLCTRL — This `ioctl()` command controls the audio output level. The SCSI command allows the control of up to four channels. The current implementation of the supported CD-ROM drive only uses channel 0 and channel 1. The valid values of volume control are between 0x00 and 0xFF, with a value of 0xFF indicating maximum volume. The third argument of the `ioctl()` call is a pointer to `struct cdrom_volctrl` which contains the output volume values.

```
/*
 * definition of audio volume control structure
 */
struct cdrom_volctrl {
    unsigned char    channel0;
    unsigned char    channel1;
    unsigned char    channel2;
    unsigned char    channel3;
};
```

The following I/O controls take a pointer that will have data returned to the user program from the CD-ROM driver.

CDROMREADTOCHDR — This `ioctl()` command returns the header of the table of contents (TOC). The header consists of the starting tracking number and the ending track number of the disc. These two numbers are returned

through a pointer of `struct cdrom_tochdr`. While the disc can start at any number, all tracks between the first and last tracks are in contiguous ascending order.

```
/*
 * definition of read toc header structure
 */
struct cdrom_tochdr {
    unsigned char    cdth_trk0;    /* starting track*/
    unsigned char    cdth_trk1;    /* ending track*/
};
```

`CDROMREADTOCENTRY` This `ioctl()` command returns the information of a specified track. The third argument of the function call is a pointer to the type `struct cdrom_tocentry`. The caller needs to supply the track number and the address format. This command will return a 4-bit `adr` field, a 4-bit `ctrl` field, the starting address in MSF format or LBA format, and the data mode if the track is a data track. The `ctrl` field specifies whether the track is data or audio.

```
/*
 * definition of read toc entry structure
 */
struct cdrom_tocentry {
    unsigned char  cdte_track;
    unsigned char  cdte_adr   :4;
    unsigned char  cdte_ctrl   :4;
    unsigned char  cdte_format;
    union {
        struct {
            unsigned char  minute;
            unsigned char  second;
            unsigned char  frame;
        } msf;
        int    lba;
    } cdte_addr;
    unsigned char  cdte_datamode;
};
```

To get the information from the leadout track, the following value is appropriate for the `cdte_track` field:

`CDROM_LEADOUT` Leadout track

To get the information from the data track, the following value is appropriate for the `cdte_ctrl` field:

`CDROM_DATA_TRACK` Data track

The following values are appropriate for the `cdte_format` field:

`CDROM_LBA` `LBA` format

`CDROM_MSF` `MSF` format

`CDROMSUBCHNL` This `ioctl()` command reads the Q sub-channel data of the current block. The subchannel data includes track number, index number, absolute `CD-ROM` address, track relative `CD-ROM` address, control data and audio status. All information is returned through a pointer to `struct cdrom_subchnl`. The caller needs to supply the address format for the returned address.

```
struct cdrom_subchnl {
    unsigned char    cdsc_format;
    unsigned char    cdsc_audiostatus;
    unsigned char    cdsc_adr:    4;
    unsigned char    cdsc_ctrl:    4;
    unsigned char    cdsc_trk;
    unsigned char    cdsc_ind;
    union {
        struct {
            unsigned char     minute;
            unsigned char     second;
            unsigned char     frame;
        } msf;
        int    lba;
    } cdsc_absaddr;
    union {
        struct {
            unsigned char     minute;
            unsigned char     second;
            unsigned char     frame;
        } msf;
        int    lba;
    } cdsc_reladdr;
};
```

The following values are valid for the audio status field returned from `READ SUBCHANNEL` command:

`CDROM_AUDIO_INVALID`	Audio status not supported.
`CDROM_AUDIO_PLAY`	Audio play operation in progress.
`CDROM_AUDIO_PAUSED`	Audio play operation paused.
`CDROM_AUDIO_COMPLETED`	Audio play successfully completed.
`CDROM_AUDIO_ERROR`	Audio play stopped due to error.

CDROM_AUDIO_NO_STATUS No current audio status to return.

CDROMREADOFFSET This `ioctl()` command returns the absolute `CD-ROM` address of the first track in the last session of a Multi-Session `CD-ROM`. The third argument of the `ioctl()` call is a pointer to an `int`.

CDROMCDDA This `ioctl()` command returns the `CD-DA` data or the subcode data. The third argument of the `ioctl()` call is a pointer to the type `struct cdrom_cdda`. In addition to allocating memory and supplying its address, the caller needs to supply the starting address of the data, the transfer length in terms of the number of blocks to be transferred, and the subcode options. The caller also needs to issue the `CDROMREADTOCENTRY ioctl()` to find out which tracks contain `CD-DA` data before issuing this `ioctl()`.

```
/*
 * Definition of CD-DA structure
 */
struct cdrom_cdda {
    unsigned int    cdda_addr;
    unsigned int    cdda_length;
    caddr_t        cdda_data;
    unsigned char    cdda_subcode;
};
```

`cdda_addr` signifies the starting logical block address.

`cdda_length` signifies the transfer length in blocks. The length of the block depends on the `cdda_subcode` selection, which is explained below.

To get the subcode information related to `CD-DA` data, the following values are appropriate for the `cdda_subcode` field:

CDROM_DA_NO_SUBCODE `CD-DA` data with no subcode.

CDROM_DA_SUBQ `CD-DA` data with sub Q code.

CDROM_DA_ALL_SUBCODE `CD-DA` data with all subcode.

CDROM_DA_SUBCODE_ONLY All subcode only.

To allocate the memory related to `CD-DA` and/or subcode data, the following values are appropriate for each data block transferred:

`CD-DA` data with no subcode 2352 bytes

`CD-DA` data with sub Q code 2368 bytes

`CD-DA` data with all subcode 2448 bytes

All subcode only	96 bytes

CDROMCDXA

This `ioctl()` command returns the CD-ROM XA (CD-ROM Extended Architecture) data according to CD-ROM XA format. The third argument of the `ioctl()` call is a pointer to the type `struct cdrom_cdxa`. In addition to allocating memory and supplying its address, the caller needs to supply the starting address of the data, the transfer length in terms of number of blocks, and the format. The caller also needs to issue the `CDROMREADTOCENTRY ioctl()` to find out which tracks contain CD-ROM XA data before issuing this `ioctl()`.

```
/*
 * Definition of CD-ROM XA structure
 */
struct cdrom_cdxa {
    unsigned int    cdxa_addr;
    unsigned int    cdxa_length;
    caddr_t        cdxa_data;
    unsigned char    cdxa_format;
};
```

To get the proper CD-ROM XA data, the following values are appropriate for the `cdxa_format` field:

`CDROM_XA_DATA`	CD-ROM XA data only
`CDROM_XA_SECTOR_DATA`	CD-ROM XA all sector data
`CDROM_XA_DATA_W_ERROR`	CD-ROM XA data with error flags data

To allocate the memory related to CD-ROM XA format, the following values are appropriate for each data block transferred:

CD-ROM XA data only	2048 bytes
CD-ROM XA all sector data	2352 bytes
CD-ROM XA data with error flags data	2646 bytes

CDROMSUBCODE

This `ioctl()` command returns raw subcode data (subcodes P ~ W are described in the "Red Book," see SEE ALSO) to the initiator while the target is playing audio. The third argument of the `ioctl()` call is a pointer to the type `struct cdrom_subcode`. The caller needs to supply the transfer length in terms of number of blocks and allocate memory for subcode data. The memory allocated should be a multiple of 96 bytes depending on the transfer length.

```
/*
 * Definition of subcode structure
 */
```

```
struct cdrom_subcode {
    unsigned int    cdsc_length;
    caddr_t        cdsc_addr;
};
```

The next group of I/O controls get and set various CD-ROM drive parameters.

CDROMGBLKMODE — This ioctl() command returns the current block size used by the CD-ROM drive. The third argument of the ioctl() call is a pointer to an integer.

CDROMSBLKMODE — This ioctl() command requests the CD-ROM drive to change from the current block size to the requested block size. The third argument of the ioctl() call is an integer which contains the requested block size.

This ioctl() command operates in exclusive-use mode only. The caller must ensure that no other processes can operate on the same CD-ROM device before issuing this ioctl(). read(2) behavior subsequent to this ioctl() remains the same: the caller is still constrained to read the raw device on block boundaries and in block multiples.

To set the proper block size, the following values are appropriate:

CDROM_BLK_512	512 bytes
CDROM_BLK_1024	1024 bytes
CDROM_BLK_2048	2048 bytes
CDROM_BLK_2056	2056 bytes
CDROM_BLK_2336	2336 bytes
CDROM_BLK_2340	2340 bytes
CDROM_BLK_2352	2352 bytes
CDROM_BLK_2368	2368 bytes
CDROM_BLK_2448	2448 bytes
CDROM_BLK_2646	2646 bytes
CDROM_BLK_2647	2647 bytes

CDROMGDRVSPEED — This ioctl() command returns the current CD-ROM drive speed. The third argument of the ioctl() call is a pointer to an integer.

CDROMSDRVSPEED — This ioctl() command requests the CD-ROM drive to change the current drive speed to the requested drive speed. This speed setting is only applicable when reading data areas. The third argument of the ioctl() is an integer which contains the requested drive speed.

To set the CD-ROM drive to the proper speed, the following values are appropriate:

CDROM_NORMAL_SPEED	150k/second
CDROM_DOUBLE_SPEED	300k/second
CDROM_QUAD_SPEED	600k/second
CDROM_MAXIMUM_SPEED	300k/second (2x drive) 600k/second (4x drive)

Note that these numbers are only accurate when reading 2048 byte blocks. The CD-ROM drive will automatically switch to normal speed when playing audio tracks and will switch back to the speed setting when accessing data.

See Also ioctl(2), read(2)

N. V. Phillips and Sony Corporation, *System Description Compact Disc Digital Audio*, ("Red Book").

N. V. Phillips and Sony Corporation, *System Description of Compact Disc Read Only Memory*, ("Yellow Book").

N. V. Phillips, Microsoft, and Sony Corporation, *System Description CD-ROM XA*, 1991.

Volume and File Structure of CD-ROM for Information Interchange, ISO 9660:1988(E).

SCSI-2 Standard, document X3T9.2/86-109

SCSI Multimedia Commands, Version 2 (MMC-2)

Notes The CDROMCDDA, CDROMCDXA, CDROMSUBCODE, CDROMGDRVSPEED, CDROMSDRVSPEED, and some of the block sizes in CDROMSBLKMODE are designed for new Sun-supported CD-ROM drives and might not work on some of the older CD-ROM drives.

CDROMCDDA, CDROMCDXA and CDROMSUBCODE will return error if the transfer length exceeds valid limits as determined appropriate. Example: for MMC-2 drives, length can not exceed 3 bytes (i.e. 0xffffff). The same restriction is enforced for older, pre-MMC-2 drives, as no limit was published for these older drives (and 3 bytes is reasonable for all media). Note that enforcing this limit does not imply that values passed in below this limit will actually be applicable for each and every piece of media.

The interface to this device is preliminary and subject to change in future releases. Programs should be written in a modular fashion so that future changes can be easily incorporated.

Name ce – Cassini Gigabit-Ethernet device driver

Synopsis `/dev/ce`

Description The `ce` Sun Gigabit-Ethernet driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface, `dlpi(7P)`, over all implementations of PCI Cassini Gigabit-Ethernet add-in adapters. Multiple Cassini-based adapters installed within the system are supported by the driver. The `ce` driver provides basic support for the Cassini-based Ethernet hardware and handles the pci108e,abba (PCI Cassini) devices. Functions include chip initialization, frame transmit and receive, multicast and promiscuous support, and error recovery and reporting. The Cassini device provides 1000BASE-SX networking interfaces using the Cassini ASIC external SERDES and fiber optical transceiver, or 10/100/1000BASE-T using a Cassini ASIC attached to a GMII twisted pair copper transceiver, or 10/100BASE-T using a Cassini ASIC attached to a MII twisted pair copper transceiver.

The 1000Base-SX standard specifies an auto-negotiation protocol to automatically select the mode of operation. In addition to the duplex mode of operation, the Cassini ASIC can auto-negotiate for *IEEE 802.3x* frame-based flow control capabilities. The Cassini PCS can perform auto-negotiation with the link's remote-end (link partner) and receives the capabilities of the remote end. It selects the highest common denominator mode of operation based on the priorities. It also supports forced-mode of operation where the driver selects the mode of operation.

Application Programming Interface The `/dev/ce` cloning character-special device is used to access all `ce` controllers installed on the system.

ce and DLPI The `ce` driver is a Style 2 data link service provider. All `M_PROTO` and `M_PCPROTO` type messages are interpreted as DLPI primitives. Valid DLPI primitives are defined in `<sys/dlpi.h>`. Refer to `dlpi(7P)` for more information. An explicit `DL_ATTACH_REQ` message by the user is required to associate the opened stream with a particular device (`ppa`). The `ppa` ID is interpreted as an `unsigned long` data type and indicates the corresponding device instance (unit) number. An error (`DL_ERROR_ACK`) is returned by the driver if the `ppa` field value does not correspond to a valid device instance number for this system. The device is initialized on first attach and de-initialized (stopped) upon last detach.

The values returned by the driver in the `DL_INFO_ACK` primitive in response to the `DL_INFO_REQ` are as follows:

- Maximum SDU is `1500` (`ETHERMTU` - defined in `<sys/ethernet.h>`).
- Minimum SDU is `0`.
- The `dlsap` address length is 8.
- MAC type is DL_ETHER.

- The sap length value is –2 meaning the physical address component is followed immediately by a 2 byte sap component within the DLSAP address.
- Service mode is DL_CLDLS.
- Optional quality of service (QOS) is not supported; the QOS fields are 0.
- Provider style is DL_STYLE2.
- Version is DL_VERSION_2.
- Broadcast address value is Ethernet/IEEE broadcast address (0xFFFFFF.)

Once in the DL_ATTACHED state, you must send a DL_BIND_REQ to associate a particular SAP (Service Access Pointer) with the stream. The ce driver interprets the sap field within the DL_BIND_REQ as an Ethernet "type," therefore valid values for the sap field are in the range [*0-0xFFFF*]. Only one Ethernet type can be bound to the stream at any time.

If you select a sap with a value of 0, the receiver will be in *802.3* mode. All frames received from the media having a "type" field in the range [*0-1500*] are assumed to be *802.3* frames and are routed up all open streams which are bound to sap value 0 . If more than one stream is in *802.3* mode, the frame will be duplicated and routed up multiple streams as DL_UNITDATA_IND messages.

In transmission, the driver checks the sap field of the DL_BIND_REQ to verify that the sap value is 0, and that the destination type field is in the range [*0-1500*]. If either is true, the driver computes the length of the message, not including initial M_PROTO mblk (message block), of all subsequent DL_UNITDATA_REQ messages and transmits *802.3* frames that have this value in the MAC frame header length field.

The ce driver DLSAP address format consists of the 6 byte physical (Ethernet) address component followed immediately by the 2 byte sap (type) component producing an 8 byte DLSAP address. Applications should *not* hard code to this particular implementation-specific DLSAP address format, but use information returned in the DL_INFO_ACK primitive to compose and decompose DLSAP addresses. The sap length, full DLSAP length, and sap/physical ordering are included within the DL_INFO_ACK. The physical address length can be computed by subtracting the sap length from the full DLSAP address length or by issuing the DL_PHYS_ADDR_REQ to obtain the current physical address associated with the stream.

Once in the DL_BOUND state, you can transmit frames on the Ethernet by sending DL_UNITDATA_REQ messages to the ce driver. The ce driver will route received Ethernet frames up all open and bound streams having a sap which matches the Ethernet type as DL_UNITDATA_IND messages. Received Ethernet frames are duplicated and routed up multiple open streams, if necessary. The DLSAP address contained within the DL_UNITDATA_REQ and DL_UNITDATA_IND messages consists of both the sap (type) and physical (Ethernet) components.

In addition to the mandatory connectionless DLPI message set, the driver additionally supports the following primitives.

ce Primitives

The DL_ENABMULTI_REQ and DL_DISABMULTI_REQ primitives enable/disable reception of individual multicast group addresses. A set of multicast addresses may be iteratively created and modified on a per-stream basis using these primitives. These primitives are accepted by the driver in any state following DL_ATTACHED.

The DL_PROMISCON_REQ and DL_PROMISCOFF_REQ primitives with the DL_PROMISC_PHYS option set in the dl_level field enables/disables reception of all "promiscuous mode" frames on the media, including frames generated by the local host. When used with the DL_PROMISC_SAP option set, this enables/disables reception of all sap (Ethernet type) values. When used with the DL_PROMISC_MULTI option set this enables/disables reception of all multicast group addresses. The effect of each is always on a per-stream basis and independent of the other sap and physical level configurations on this stream or other streams.

The DL_PHYS_ADDR_REQ primitive returns the 6 octet Ethernet address currently associated (attached) to the stream in the DL_PHYS_ADDR_ACK primitive. This primitive is valid only in states following a successful DL_ATTACH_REQ.

The DL_SET_PHYS_ADDR_REQ primitive changes the 6 octet Ethernet address currently associated (attached) to this stream. The credentials of the process which originally opened this stream must be superuser. Otherwise EPERM is returned in the DL_ERROR_ACK. This primitive is destructive because it affects all other current and future streams attached to this device. An M_ERROR is sent up all other streams attached to this device when this primitive is successful on this stream. Once changed, all streams subsequently opened and attached to this device will obtain this new physical address. Once changed, the physical address will remain until this primitive is used to change the physical address again or the system is rebooted, whichever comes first.

ce DRIVER

By default, the ce driver performs auto-negotiation to select the mode and flow control capabilities of the link.

The link can assume one of the following modes:

- 1000 Mbps, full-duplex
- 1000 Mbps, half-duplex
- Symmetric pause
- Asymmetric pause

Speeds and modes are described in the 1000Base-TX standard.

The auto–negotiation protocol automatically selects:

- Operation mode (half-duplex or full-duplex)
- Flow control capability (symmetric and/or asymmetric)

The auto–negotiation protocol does the following:

- Gets all modes of operation supported by the link partner.
- Advertises its capabilities to the link partner.
- Selects the highest common denominator mode of operation based on the priorities.

The Cassini hardware can operate in all modes listed above, providing auto-negotiation is used by default to bring up the link and select the common mode of operation with the link partner. The PCS also supports forced-mode of operation in which the driver can select the mode of operation and the flow control capabilities, using the `ndd` utility.

The Cassini device also supports programmable IPG (Inter-Packet Gap) parameters ipg1 and ipg2. By default, the driver sets ipg1 and ipg2 to 8 and 4 byte-times respectively (which are the standard values). If desired, you can alter these values from the standard 1000 Mpbs IPG set to 0.096 microseconds.

ce Parameter List

The `ce` driver enables the setting and getting of various parameters for the Cassini device. The parameter list includes *current transceiver status*, *current link status*, *inter-packet gap*, *PCS capabilities* and *link partner capabilities*.

The PCS features two set of capabilities. One set reflects the capabilities of the hardware and are read-only. The second set, which reflects the values you choose, are used in speed selection and possess read/write capabilities. At boot time, these two sets of capabilities are the same. The link partner capabilities are also read-only because the current default value of these parameters can be read but not modified.

Files

`/dev/ce`	ce special character device.
`/platform/sun4u/kernel/drv/ce.conf`	SPARC system-wide default device driver properties.
`/platform/amd64/kernel/drv/ce.conf`	64–bit x86 system-wide default device driver properties.
`/kernel/drv/ce.conf`	32–bit x86 system-wide default device driver properties.
`/kernel/drv/amd64/ce.conf`	64–bit x86 system-wide default device driver properties.

See Also

`ndd`(1M), `netstat`(1M), `driver.conf`(4), `ge`(7D), `hme`(7D), `qfe`(7d), `dlpi`(7P)

Name chxge – Chelsio Ethernet network interface controllers

Synopsis `/dev/chxge`

Description The `chxge` Ethernet driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface, `dlpi(7P)`, over Chelsio NIC controllers. Multiple (and mixed) NIC controllers installed within the system are supported by the driver. The `chxge` driver provides basic support for the NIC hardware. Functions include chip initialization, frame transmit and receive, and error recovery and reporting.

Application Programming Interface The cloning, character-special device `/dev/chxge` is used to access NIC devices installed within the system.

chxge and Dlpi The `chxge` driver is dependent on `/kernel/misc/gld`, a loadable kernel module that provides the `chxge` driver with the DLPI and STREAMS functionality required of a LAN driver. See `gld(7D)` for more details on the primitives supported by the driver.

The values returned by the driver in the DL_INFO_ACK primitive in response to the DL_INFO_REQ are:

- Default Maximum SDU is 1500 (ETHERMTU).
- dlsap address length is 8.
- MAC type is DL_ETHER.
- The sap length value is -2, meaning the physical address component is followed immediately by a 2-byte sap component within the DLSAP address.
- Broadcast address value is Ethernet/IEEE broadcast address (FF:FF:FF:FF:FF:FF).

Files

/dev/chxge	Character special device.
/kernel/drv/sparcv9/chxge	SPARC chxge driver binary.
/kernel/drv/chxge	x86 platform kernel module. (32-bit).
/kernel/drv/amd64/chxge	x86 platform kernel module. (64-bit).

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86

See Also `netstat(1M)`, `attributes(5)`, `gld(7D)`, `dlpi(7P)`, `gld(9F)`, `gld_mac_info(9S)`

Name cmdk – common disk driver

Synopsis `cmdk@`*target*`,` *lun* `:` `[` *partition* `|` *slice* `]`

Description The `cmdk` device driver is a common interface to various disk devices. The driver supports magnetic fixed disks and magnetic removable disks.

The `cmdk` device driver supports three different disk labels: fdisk partition table, Solaris x86 VTOC and EFI/GPT.

The block-files access the disk using the system's normal buffering mechanism and are read and written without regard to physical disk records. There is also a "raw" interface that provides for direct transmission between the disk and the user's read or write buffer. A single read or write call usually results in one I/O operation; raw I/O is therefore considerably more efficient when many bytes are transmitted. The names of the block files are found in `/dev/dsk`. Raw file names are found in `/dev/rdsk`.

I/O requests to the magnetic disk must have an offset and transfer length that is a multiple of 512 bytes or the driver returns an `EINVAL` error.

Slice 0 is normally used for the root file system on a disk, slice 1 as a paging area (for example, swap), and slice 2 for backing up the entire fdisk partition for Solaris software. Other slices may be used for `usr` file systems or system reserved area.

The fdisk partition 0 is to access the entire disk and is generally used by the `fdisk(1M)` program.

Files

`/dev/dsk/c`*n*`d`*n*`[s|p]`*n*	block device (IDE)
`/dev/rdsk/c`*n*`d`*n*`[s|p]`*n*	raw device (IDE)
	where:
	`c`*n* controller *n*.
	`d`*n* lun *n (0-1)*.
	`s`*n* UNIX system slice *n (0-15)*.
	`p`*n* fdisk partition (0–36).
`/kernel/drv/cmdk`	32-bit kernel module.
`/kernel/drv/amd64/cmdk`	64-bit kernel module.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86

See Also fdisk(1M), mount(1M), lseek(2), read(2), write(2), readdir(3C), scsi(4), vfstab(4), attributes(5), dkio(7I)

Name connld – line discipline for unique stream connections

Synopsis

```
#include </sys/steam.h>

int ioctl(fd,I_PUSH,"connld");
```

Description `connld` is a STREAMS-based module that provides unique connections between server and client processes. It can only be pushed (see `streamio(7I)`) onto one end of a STREAMS-based pipe that may subsequently be attached to a name in the file system name space with `fattach(3C)`. After the pipe end is attached, a new pipe is created internally when an originating process attempts to `open(2)` or `creat(2)` the file system name. A file descriptor for one end of the new pipe is packaged into a message identical to that for the ioctl `I_SENDFD` (see `streamio(7I)`) and is transmitted along the stream to the server process on the other end. The originating process is blocked until the server responds.

The server responds to the `I_SENDFD` request by accepting the file descriptor through the `I_RECVFD` ioctl message. When this happens, the file descriptor associated with the other end of the new pipe is transmitted to the originating process as the file descriptor returned from `open(2)` or `creat(2)`.

If the server does not respond to the `I_SENDFD` request, the stream that the `connld` module is pushed on becomes uni-directional because the server will not be able to retrieve any data off the stream until the `I_RECVFD` request is issued. If the server process exits before issuing the `I_RECVFD` request, the `open(2)` or the `creat(2)` invocation will fail and return -1 to the originating process.

When the `connld` module is pushed onto a pipe, it ignores messages going back and forth through the pipe.

Errors On success, an open of `connld` returns 0. On failure, `errno` is set to the following values:

`EINVAL` A stream onto which `connld` is being pushed is not a pipe or the pipe does not have a write queue pointer pointing to a stream head read queue.

`EINVAL` The other end of the pipe onto which `connld` is being pushed is linked under a multiplexor.

`EPIPE` `connld` is being pushed onto a pipe end whose other end is no longer there.

`ENOMEM` An internal pipe could not be created.

`ENXIO` An `M_HANGUP` message is at the stream head of the pipe onto which `connld` is being pushed.

`EAGAIN` Internal data structures could not be allocated.

`ENFILE` A file table entry could not be allocated.

See Also creat(2), open(2), fattach(3C), streamio(7I)

STREAMS Programming Guide

Name console – STREAMS-based console interface

Synopsis `/dev/console`

Description The file `/dev/console` refers to the system console device. `/dev/console` should be used for interactive purposes only. Use of `/dev/console` for logging purposes is discouraged; `syslog(3C)` or `msglog(7D)` should be used instead.

The identity of this device depends on the EEPROM or NVRAM settings in effect at the most recent system reboot; by default, it is the "workstation console" device consisting of the workstation keyboard and frame buffer acting in concert to emulate an ASCII terminal (see `wscons(7D)`).

Regardless of the system configuration, the console device provides asynchronous serial driver semantics so that, in conjunction with the STREAMS line discipline module `ldterm(7M)`, it supports the `termio(7I)` terminal interface.

See Also `syslog(3C)`, `termios(3C)`, `ldterm(7M)`, `termio(7I)`, `msglog(7D)`, `wscons(7D)`

Notes In contrast to pre-SunOS 5.0 releases, it is no longer possible to redirect I/O intended for `/dev/console` to some other device. Instead, redirection now applies to the workstation console device using a revised programming interface (see `wscons(7D)`). Since the system console is normally configured to be the work station console, the overall effect is largely unchanged from previous releases.

See `wscons(7D)` for detailed descriptions of control sequence syntax, ANSI control functions, control character functions and escape sequence functions.

Name cpqary3 – provides disk and SCSI tape support for HP Smart Array controllers

Description The `cpqary3` module provides low-level interface routines between the common disk I/O subsystem and the HP SMART Array controllers. The `cpqary3` driver provides disk and SCSI tape support for the HP Smart Array controllers.

Please refer to the *cpqary3 Release Notes*, for the supported HP Smart Array Controllers and Storage boxes.

Each of the controller should be the sole initiator on a SCSI bus. Auto configuration code determines if the adapter is present at the Configured address and what types of devices are attached to it.

Configuration Use the Array Configuration Utility to configure the controllers. Each controller can support up to 32 logical volumes. In addition, each controller supports up to a maximum of 28 connected SCSI tape drives. With 1.90 and later versions of `cpqary3` driver, HP Smart Array SAS controllers, having Firmware Revision 5.10 or later, support 64 logical drives. This firmware also supports Dual Domian Multipath configurations.

The driver attempts to initialize itself in accordance with the information found in the configuration file, `/kernel/drv/cpqary3.conf`.

The target driver's configuration file need entries if support is needed for targets numbering greater than the default number of targets supported by the corresponding target driver.

By default, entries for SCSI target numbers 0 to 15 are present in `sd.conf`. Entries for target numbers 16 and above are added in SCSI class in the `sd.conf` file for supporting corresponding logical volumes.

If SCSI tape drives are connected to the supported controllers, entries for target IDs from `33` to `33+`*n* must be added in the `/kernel/drv/st.conf` file under `scsi` class, where *n* is the total number of SCSI tape drives connected to the controller with largest number of tape drives connected to it, in the existing configuration. For example, two supported controllers, `c1` and `c2` are present in the system. If controller `c1` has two tape drives and controller `c2` has five tape drives connected, entries for target IDs `33` through `38` are required under `scsi` class in `/kernel/drv/st.conf` file. The maximum number of tape drives that can be connected to a controller is `28`. With 1.90 and later versions of the `cpqary3` driver, if tape drives are connected to the Smart Array SAS controllers, target ID entries for tape drives from `65` to `65+`*n* must be added in the `/kernel/drv/st.conf` file under the `scsi` class.

Files

`/kernel/drv/cpqary3.conf`	Configuration file for `CPQary3`
`/kernel/drv/sd.conf`	Configuration file for `sd`
`/kernel/drv/st.conf`	Configuration file for `st`
`/dev/dsk`	Block special file names for disk device

`/dev/rdsk`	Character special file names for disk device
`/dev/rmt`	Special file names for SCSI tape devices

See Also `driver.conf`(4), `sd`(7D), `st`(7D)

cpqary3 Release Notes

Notes The Smart Array controllers supported by the current version of the `cpqary3` driver do not support `format unit` SCSI command. Therefore, selecting the `format` option under the `format` utility main menu is not supported. In addition, the `repair` option under `format` utility main menu is not supported as this operation is not applicable to Logical volumes connected to the supported Smart Array controllers.

The names of the block files can be found in `/dev/dsk`. The names of the raw files can be found in `/dev/rdsk`.

Name cpr – Suspend and resume module

Synopsis

```
/platform/'uname -m'/kernel/misc/cpr
```

Description The `cpr` module is a loadable module used to suspend and resume the entire system. You may wish to suspend a system to save power or to power off temporarily for transport. The `cpr` module should not be used in place of a normal shutdown when performing any hardware reconfiguration or replacement. In order for the resume operation to succeed, it is important that the hardware configuration remain the same. When the system is suspended, the entire system state is preserved in non-volatile storage until a resume operation is conducted.

`dtpower`(1M) or `power.conf`(4) are used to configure the suspend-resume feature.

The speed of suspend and resume operations can range from 15 seconds to several minutes, depending on the system speed, memory size, and load.

During resume operation, the `SIGTHAW` signal is sent to all processes to allow them to do any special processing in response to suspend-resume operation. Normally applications are not required to do any special processing because of suspend-resume, but some specialized processes can use `SIGTHAW` to restore the state prior to suspend. For example, X can refresh the screen in response to `SIGTHAW`.

In some cases the `cpr` module may be unable to perform the suspend operation. If a system contains additional devices outside the standard shipped configuration, it is possible that device drivers for these additional devices might not support suspend-resume operations. In this case, the suspend fails and an error message is displayed. These devices must be removed or their device drivers unloaded for the suspend operation to succeed. Contact the device manufacturer to obtain a new version of device driver that supports suspend-resume.

A suspend may also fail when devices or processes are performing critical or time-sensitive operations (such as realtime operations). The system will remain in its current running state. Messages reporting the failure will be displayed on the console and status returned to the caller. Once the system is successfully suspended the resume operation will succeed, barring external influences such as a hardware reconfiguration.

Some network-based applications may fail across a suspend and resume cycle. This largely depends on the underlying network protocol and the applications involved. In general, applications that retry and automatically reestablish connections will continue to operate transparently on a resume operation; those applications that do not will likely fail.

Attributes See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcpr

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface stability	Unstable

See Also `dtpower`(1M) (OpenWindows Reference Manual), `pmconfig`(1M), `uadmin`(1M), `uadmin`(2), `power.conf`(4), `attributes`(5)

Using Power Management

Writing Device Drivers

Notes Certain device operations such as tape and floppy disk activities are not resumable due to the nature of removable media. These activities are detected at suspend time, and must be stopped before the suspend operation will complete successfully.

Suspend-resume is currently supported only on a limited set of hardware platforms. Please see the book *Using Power Management* for a complete list of platforms that support system Power Management. See `uname`(2) to programatically determine if the machine supports suspend-resume.

Name cpuid – CPU identification driver

Synopsis

```
/dev/cpu/self/cpuid
```

Description

SPARC and x86 system

This device provides implementation-private information via ioctls about various aspects of the implementation to Solaris libraries and utilities.

x86 systems only

This device also provides a file-like view of the namespace and return values of the x86 cpuid instruction. The cpuid instruction takes a single 32-bit integer function code, and returns four 32-bit integer values corresponding to the input value that describe various aspects of the capabilities and configuration of the processor.

The API for the character device consists of using the seek offset to set the function code value, and using a `read(2)` or `pread(2)` of 16 bytes to fetch the four 32-bit return values of the instruction in the order `%eax`, `%ebx`, `%ecx` and `%edx`.

No data can be written to the device. Like the `cpuid` instruction, no special privileges are required to use the device.

The device is useful to enable low-level configuration information to be extracted from the CPU without having to write any assembler code to invoke the `cpuid` instruction directly. It also allows the kernel to attempt to correct any erroneous data returned by the instruction (prompted by occassional errors in the information exported by various processor implementations over the years).

See the processor manufacturers documentation for further information about the syntax and semantics of the wide variety of information available from this instruction.

Example This example allows you to determine if the current x86 processor supports "long mode," which is a necessary (but not sufficient) condition for running the 64-bit Solaris kernel on the processor.

```
/*

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <stdio.h>

static const char devname[] = "/dev/cpu/self/cpuid";

/*ARGSUSED*/
int
main(int argc, char *argv[])
```

```
{
        struct {
                uint32_t r_eax, r_ebx, r_ecx, r_edx;
        } _r, *rp = &_r;
        int d;
        char *s;

        if ((d = open(devname, O_RDONLY)) == -1) {
                perror(devname);
                return (1);
        }

        if (pread(d, rp, sizeof (*rp), 0) != sizeof (*rp)) {
                perror(devname);
                goto fail;
        }

        s = (char *)&rp->r_ebx;
        if (strncmp(s, "Auth" "cAMD" "enti", 12) != 0 &&
            strncmp(s, "Genu" "ntel" "ineI", 12) != 0)
                goto fail;

        if (pread(d, rp, sizeof (*rp), 0x80000001) == sizeof (*rp)) {
                /*
                 * Read extended feature word; check bit 29
                 */
                (void) close(d);
                if ((rp->r_edx >> 29) & 1) {
                        (void) printf("processor supports long mode\n");
                        return (0);
                }
        }
fail:
        (void) close(d);
        return (1);
}
```

Errors

ENXIO	Results from attempting to read data from the device on a system that does not support the CPU identification interfaces
EINVAL	Results from reading from an offset larger than UINT_MAX, or attempting to read with a size that is not multiple of 16 bytes.

Files

`/dev/cpu/self/cpuid`	Provides access to CPU identification data.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWckr
Interface Stability	Evolving

See Also psrinfo(1M), prtconf(1M), pread(2), read(2), attributes(5)

Name ctfs – contract file system

Description The `ctfs` filesystem is the interface to the contract sub-system. `ctfs` is mounted during boot at `/system/contract`. For information on contracts and the contents of this filesystem, see `contract(4)`.

Files `/system/contract` Mount point for the `ctfs` file system

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWckr

See Also `contract(4)`, `vfstab(4)`, `attributes(5)`, `smf(5)`

Name ctsmc – System Management Controller driver

Description The `ctsmc` system management controller driver is a multithreaded, loadable, clonable STREAMS hardware driver that supports communication with the system management controller device on SUNW,NetraCT-410, SUNW,NetraCT-810 and SUNW,Netra-CP2300 platforms.

The `smc` device provides a Keyboard Controller Style (KCS) interface as described in the *Intelligent Platform Management Interface (IPMI) Version 1.5* specification. The `ctsmc` driver enables user-land and kernel-land clients to access services provided by `smc` hardware.

Files

`/dev/ctsmc`	`ctsmc` special character device
`/platform/sun4u/kernel/drv/sparcv9/ctsmc`	64 bit ELF kernel driver

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC
Availability	SUNWcar.u

See Also `attributes(5)`

STREAMS Programmers Guide

Writing Device Drivers

Intelligent Platform Management Interface (IPMI). Version 1.5 – PICMIG, February, 2001

Name cvc – virtual console driver

Description The cvc virtual console driver is a STREAMS-based pseudo driver that supports the network console. The cvc driver interfaces with console(7D).

Logically, the cvc driver sits below the console driver. It redirects console output to the cvcredir(7D) driver if a network console connection is active. If a network console connection is not active, it redirects console output to an internal hardware interface.

The cvc driver receives console input from cvcredir and internal hardware and passes it to the process associated with /dev/console.

Notes The cvc facility supersedes the SunOS wscons(7D) facility, which should *not* be used in conjunction with cvc. The wscons driver is useful for systems with directly attached consoles (frame buffers and keyboards), but is not useful with platforms using cvc, which have no local keyboard or frame buffer.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	Sun Enterprise 10000 servers, Sun Fire 15000 servers
Availability	SUNWcvc.u

See Also cvcd(1M), attributes(5), console(7D), cvcredir(7D), wscons(7D)

Sun Enterprise 10000 SSP Reference Manual

Sun System Management Services (SMS) Reference Manual

Name cvcredir – virtual console redirection driver

Description The `cvcredir` virtual console redirection driver is a STREAMS-based pseudo driver that supports the network console provided on some platforms. The `cvcredir` driver interfaces with the virtual console driver `cvc`(7D), and the virtual console daemon, `cvcd`(1M).

The `cvcredir` driver receives console output from `cvc` and passes it to `cvcd`. It receives console input from `cvcd` and passes it to `cvc`.

Attributes See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	Sun Enterprise 10000 servers, Sun Fire 15K servers
Availability	SUNWcvc.u

See Also `cvcd`(1M), `attributes`(5), `console`(7D), `cvc`(7D)

Sun Enterprise 10000 SSP Reference Manual

Sun System Management Services (SMS) Reference Manual

Name dad – driver for IDE disk devices

Synopsis `dad@` *target*`,`*lun*`:`*partition*

Description This driver handles the ide disk drives on SPARC platforms. The type of disk drive is determined using the ATA IDE identify device command and by reading the volume label stored on the drive. The `dad` device driver supports the Solaris SPARC VTOC and the EFI/GPT disk volume labels.

The block-files access the disk using the system's normal buffering mechanism and are read and written without regard to physical disk records. There is also a "raw" interface that provides for direct transmission between the disk and the user's read or write buffer. A single read or write call usually results in one I/O operation; raw I/O is therefore considerably more efficient when many bytes are transmitted. The names of the block files are found in `/dev/dsk`. Raw file names are found in `/dev/rdsk`.

I/O requests to the raw device must be aligned on a 512-byte (`DEV_BSIZE`) boundary and must have a length that is a multiple of 512 bytes. Requests that do not meet the restrictions cause the driver to return an `EINVAL` error. I/O requests to the block device have no alignment or length restrictions.

Device Statistics Support

Each device maintains I/O statistics both for the device and for each partition allocated on that device. For each device/partition, the driver accumulates reads, writes, bytes read, and bytes written. The driver also takes hi-resolution time stamps at queue entry and exit points, which facilitates monitoring the residence time and cumulative residence-length product for each queue.

Each device also has error statistics associated with it. These must include counters for hard errors, soft errors and transport errors. Other data may be implemented as required.

Files

`/dev/dsk/c`*n*`t`*n*`d`*n*`s`*n* block files

`/dev/rdsk/c`*n*`t`*n*`d`*n*`s`*n* raw files

where:

`c`*n* controller *n*

`t`*n* IDE target id *n* (0-3)

`d`*n* Always 0.

`s`*n* partition *n* (0-7)

The target ide numbers are assigned as:

0 Master disk on Primary channel.

1 Slave disk on Primary channel.

2 Master disk on Secondary channel

3 Slave disk on Secondary channel.

ioctls Refer to dkio(7I).

Errors

EACCES	Permission denied.
EBUSY	The partition was opened exclusively by another thread.
EFAULT	Argument was a bad address.
EINVAL	Invalid argument.
EIO	I/O error occurred.
ENOTTY	The device does not support the requested ioctl function.
ENXIO	The device did not exist during opening.
EROFS	The device is a read-only device.

See Also format(1M), mount(1M), lseek(2), read(2), write(2), driver.conf(4), vfstab(4), dkio(7I)

X3T10 ATA-4 specifications.

Diagnostics

Command:<*number*>, Error:<*number*>, Status:<*number*>
Indicates that the command failed with an error and provides status register contents. Where <*number*> is a hexadecimal value.

offline
The driver has decided that the target disk is no longer there.

disk ok
The target disk is now responding again.

disk not responding to selection
The target disk is not responding.

i/o to invalid geometry
The geometry of the drive could not be established.

incomplete read/write - retrying/giving up
There was a residue after the command completed normally.

no bp for disk label
A bp with consistent memory could not be allocated.

no memory for disk label
Free memory pool exhausted.

ATA transport failed: reason 'nnnn': {retrying|giving}
The host adapter has failed to transport a command to the target for the reason stated. The driver will either retry the command or, ultimately, give up.

`no mem for property`
Free memory pool exhausted.

`transport rejected (<n>)`
Host adapter driver was unable to accept a command.

`Device Fault`
Device fault - reason for such error is vendor specific.

Name daplt – Tavor uDAPL service driver

Synopsis `daplt@0:daplt`

Description The `daplt` module is the driver component of the uDAPL service provider for Tavor which implements the provider functions of the *uDAPL Specification 1.2* described under `libdat(3LIB)`.

The `daplt` module is a child of the IB nexus driver, `ib(7D)`, and layers on top of the Solaris kernel IB Transport Layer, `ibtl(7D)`. The `daplt` driver uses the InfiniBand Transport Framework (IBTF). (See `ibtl(7D)`, `ibcm(7D)`, and `ib(7D)` to access privileged IB VERBS.)

The `daplt` driver copies out various HCA H/W object reference handles, including working and completion queues and User Access Region registers, to its own uDAPL service provider library for Tavor. The library can refer back to these object handles and use them to `mmap(2)` in the mapping of these H/W queues and registers from the Tavor HCA driver, `tavor(7D)`. This process enables time-critical non-privileged IB VERBS such as send/receive work elements, RDMA read/write and memory window bind, to be invoked in the userland library and performed directly by the firmware or hardware. As a result, OS and network stack are bypassed, achieving true zero data copy with the lowest possible latency.

Files

`/kernel/drv/sparcv9/daplt`	64-bit SPARC ELF kernel driver
`/kernel/drv/daplt`	32–bit x86 ELF kernel driver
`/kernel/drv/amd64/daplt`	64–bit x86 ELF kernel driver
`/kernel/drv/daplt.conf`	driver configuration file
`/dev/daplt`	special character device.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86
Availability	SUNWdapltr

See Also `mmap(2)`, `libdat(3LIB)`, `driver.conf(4)`, `attributes(5)`, `ib(7D)`, `ibcm(7D)`, `ibdm(7D)`, `ibtl(7D)`, `tavor(7D)`

uDAPL Specification 1.2

Name dca – Crypto Accelerator device driver

Synopsis

`pci108e,5454@`*pci-slot*

`pci108e,5455@`*pci-slot*

`pci108e,5456@`*pci-slot*

`pci14e4,5820@`*pci-slot*

`pci14e4,5821@`*pci-slot*

`pci14e4,5822@`*pci-slot*

Description The `dca` device driver is a multi-threaded, loadable hardware driver supporting Sun PCI-based (`pci108e,5454`) cryptographic accelerators, such as the Sun Crypto Accelerator 1000.

The `dca` driver requires the presence of Solaris Cryptographic Framework for applications and kernel clients to access the provided services.

Extended Description The `dca` driver maintains the following statistics:

`3desjobs`	Total number of jobs submitted to the device for 3DES encryption.
`3desbytes`	Total number of bytes submitted to the device for 3DES encryption.
`rsapublic`	Total number of jobs submitted to the device for RSA public key operations.
`rsaprivate`	Total number of jobs submitted to the device for RSA private key operations.
`dsasign`	Total number of jobs submitted to the device for DSA signing.
`dsaverify`	Total number of jobs submitted to the device for DSA verification.
`rngjobs`	Total number of jobs submitted for pure entropy generation.
`rngbytes`	Total number of bytes of pure entropy requested from the device.
`rngsha1jobs`	Total number of jobs submitted for entropy generation, with SHA-1 post-processing.
`rngsha1bytes`	Total number of bytes of entropy requested from the device, with SHA-1 post-processing.

Additional statistics may be supplied for Sun support personnel, but are not useful to end users and are not documented here.

The `dca` driver can be configured by defining properties in `/kernel/drv/dca.conf` which override the default settings. The following properties are supported:

`nostats` Disables the generation of statistics. This property may be used to help prevent traffic analysis, but this may inhibit support personnel.

`rngdirect` Disables the SHA-1 post-processing of generated entropy. This may give "truer" random numbers, but it may also introduce the risk of external biases influencing the distribution of generated random numbers.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC
Availability	SUNWdcar
Interface Stability	Unstable

Files

`/kernel/drv/dca.conf`	`dca` configuration file
`/kernel/drv/sparcv9/dca`	64–bit ELF kernel driver (SPARC)
`/kernel/drv/dca`	32–bit ELF kernel driver (x86)
`/kernel/drv/amd64/dca`	64–bit ELF kernel driver (AMD64)

See Also `cryptoadm(1M)`, `kstat(1M)`, `prtconf(1M)`, `driver.conf(4)`, `attributes(5)`

Solaris Cryptographic Framework.

Name dcam1394 – 1394–based digital camera (IIDC) driver

Synopsis

```
#include <sys/dcam/dcam1394_io.h>
```

Description The `dcam1394` driver supports devices implementing the *1394 Trade Association Digital Camera Specification* (also referred to as the IIDC specification). Only a subset of the specification is supported.

Reading Data Isochronous data is read from the driver frame-by-frame and is maintained within the driver in a ring buffer.

Video frames are read from the isochronous input device using `read(2)`.

The `dcam1394_frame_t` structure describes the frame layout and is defined as follows:

```
struct {
     unsigned int vid_mode;
     unsigned int seq_num;
     hrtime_t     timestamp;
     unsigned char *buff;
};
```

The size to allocate for the structure is determined by the video mode for which the camera is configured. Possible values for the vid_mode field are listed under DCAM1394_PARAM_VID_MODE below.

Ioctl Requests The following ioctl requests are supported:

DCAM1394_CMD_CAM_RESET
Reset the device.

DCAM1394_CMD_REG_READ
Read the indicated dcam/IIDC register. The argument is a pointer to a `dcam1394_reg_io_t` structure, which is defined as follows:

```
struct {
     unsigned int offs;
     unsigned int val;
};
```

The offs field should be set to the offset of the register from which to read. Register offset values are defined in the *1394 Trade Association Digital Camera Specification*.

After the operation is completed, the camera register value is put in the `val` field.

DCAM1394_CMD_REG_WRITE
Write the indicated dcam/IIDC register. The argument is a pointer to a `dcam1394_reg_io_t` structure (described above).

The offs field should be set to the offset of the register from which to read. The register offset values are defined in the *1394 Trade Association Digital Camera Specification*.

The val field should be set to the value to be written to the camera register.

DCAM1394_CMD_PARAM_GET
Gets a list of parameters associated with a camera. The argument is a pointer to a `dcam1394_param_list_t` structure (described below). The parameter list is accessed through macros defined below.

The paramter list only supports Format 1 video formats.

DCAM1394_CMD_PARAM_SET
Sets a list of parameters associated with a camera. The argument is a pointer to a `dcam1394_param_list_t structure` (described below). The parameter list is accessed through macros defined below.

The paramter list only supports Format 1 video formats.

DCAM1394_CMD_FRAME_RCV_START
Start receiving video frames from the camera.

The contents of the ring buffer may be accessed by reading the isochronous stream device.

DCAM1394_CMD_FRAME_RCV_STOP
Stop receiving frames from the camera.

DCAM1394_CMD_RING_BUFF_FLUSH
Flush the frames in the ring buffer.

DCAM1394_CMD_FRAME_SEQ_NUM_COUNT_RESET
Reset frame sequence number.

Parameter List Access

The parameter list is initialized and access through macros. The data type for the parameter list is `dcam1394_param_list_t`.

The following macros are used to access the parameter list:

PARAM_LIST_INIT(param_list)
Initialize the parameter list.

PARAM_LIST_ADD(param_list, param, subparam)
Add a parameter to the list.

PARAM_LIST_REMOVE(param_list, param, subparam)
Remove a parameter from the list.

PARAM_LIST_IS_ENTRY(param_list, param, subparam)
Indicates if a specific parameter is in the list.

PARAM_VAL(param_list, param, subparam)
Value of a specified parameter.

PARAM_ERR(param_list, param, subparam)
Indicates if a specific parameter is successfully set.

When no subparam value is required, the value DCAM1394_SUBPARAM_NONE may be used.

Parameters The following parameters may appear in the list:

DCAM1394_PARAM_CAP_POWER_CTRL
Queries if the camera can be turned off and on through software. The subparam value is ignored.

DCAM1394_PARAM_POWER
Controls or queries if the camera is powered up. Verify this feature using DCAM1394_PARAM_CAP_POWER_CTRL before use. The subparam field is ignored.

DCAM1394_PARAM_CAP_VID_MOD
Queries if a specific video mode is supported by the camera.

subparam is one of the following and is used to determine if a specified video mode is supported by the camera:

```
DCAM1394_SUBPARAM_VID_MODE_0
DCAM1394_SUBPARAM_VID_MODE_YUV_444_160_120
     Video mode is 4:4:4, YUV color space, 160x120 resolution.

DCAM1394_SUBPARAM_VID_MODE_1
DCAM1394_SUBPARAM_VID_MODE_YUV_422_320_240
     Video mode is 4:2:2, YUV color space, 320x240 resolution.

DCAM1394_SUBPARAM_VID_MODE_2
DCAM1394_SUBPARAM_VID_MODE_YUV_411_640_480
     Video mode is 4:1:1, YUV color space, 640x480 resolution.

DCAM1394_SUBPARAM_VID_MODE_3
DCAM1394_SUBPARAM_VID_MODE_YUV_422_640_480
     Video mode is 4:2:2, YUV color space, 640x480 resolution.

DCAM1394_SUBPARAM_VID_MODE_4
DCAM1394_SUBPARAM_VID_MODE_RGB_640_480
     Video mode is RGB color space, 640x480 resolution.

DCAM1394_SUBPARAM_VID_MODE_5
DCAM1394_SUBPARAM_VID_MODE_Y_640_480
     Video mode is Y color space, 640x480 resolution.
```

DCAM1394_PARAM_VID_MODE
Controls or queries the current video mode of the camera. The subparam field is ignored. When selecting the video mode, it should be compatible with the capability of the camera, which may be determined by checking the DCAM1394_PARAM_CAP_VID_MODE parameter.

The value of this parameter may be one of the following:

```
DCAM1394_VID_MODE_0
DCAM1394_VID_MODE_YUV_444_160_120
     Video mode is 4:4:4, YUV color space, 160x120 resolution.

DCAM1394_VID_MODE_1
DCAM1394_VID_MODE_YUV_422_320_240
     Video mode is 4:2:2, YUV color space, 320x240 resolution.

DCAM1394_VID_MODE_2
DCAM1394_VID_MODE_YUV_411_640_480
     Video mode is 4:1:1, YUV color space, 640x480 resolution.

DCAM1394_VID_MODE_3
DCAM1394_VID_MODE_YUV_422_640_480
     Video mode is 4:2:2, YUV color space, 640x480 resolution.

DCAM1394_VID_MODE_4
DCAM1394_VID_MODE_RGB_640_480
     Video mode is RGB color space, 640x480 resolution.

DCAM1394_VID_MODE_5
DCAM1394_VID_MODE_Y_640_480
     Video mode is Y color space, 640x480 resolution.
```

DCAM1394_PARAM_CAP_FRAME_RATE_VID_MODE_0

Queries if a specific frame rate is supported by the camera in video mode 0 (4:4:4, YUV, 160x120).

subparam is one of the following and used to determine if a specified frame rate is supported by the camera:

```
DCAM1394_SUBPARAM_FRAME_RATE_0
DCAM1394_SUBPARAM_FRAME_RATE_3_75_FPS
     Frame rate is 3.75 frames/second.

DCAM1394_SUBPARAM_FRAME_RATE_1
DCAM1394_SUBPARAM_FRAME_RATE_7_5_FPS
     Frame rate is 7.5 frames/second.

DCAM1394_SUBPARAM_FRAME_RATE_2
DCAM1394_SUBPARAM_FRAME_RATE_15_FPS
     Frame rate is 15 frames/second.

DCAM1394_SUBPARAM_FRAME_RATE_3
DCAM1394_SUBPARAM_FRAME_RATE_30_FPS
     Frame rate is 30 frames/second.

DCAM1394_SUBPARAM_FRAME_RATE_4
DCAM1394_SUBPARAM_FRAME_RATE_60_FPS
     Frame rate is 60 frames/second.
```

DCAM1394_PARAM_CAP_FRAME_RATE_VID_MODE_1
Queries if a specific frame rate is supported by the camera in video mode 1 (4:2:2, YUV, 320x240). See DCAM1394_PARAM_CAP_FRAME_RATE_VID_MODE_0 for a listing of valid subparam values.

DCAM1394_PARAM_CAP_FRAME_RATE_VID_MODE_2
Queries if a specific frame rate is supported by the camera in video mode 2 (4:1:1, YUV, 640x480). See DCAM1394_PARAM_CAP_FRAME_RATE_VID_MODE_0 for a listing of valid subparam values.

DCAM1394_PARAM_CAP_FRAME_RATE_VID_MODE_3
Queries if a specific frame rate is supported by the camera in video mode 3 (4:2:2, YUV, 640x480). See DCAM1394_PARAM_CAP_FRAME_RATE_VID_MODE_0 for a listing of valid subparam values.

DCAM1394_PARAM_CAP_FRAME_RATE_VID_MODE_4
Queries if a specific frame rate is supported by the camera in video mode 4. (RGB, 640x480). See DCAM1394_PARAM_CAP_FRAME_RATE_VID_MODE_0 for a listing of valid subparam values.

DCAM1394_PARAM_CAP_FRAME_RATE_VID_MODE_5
Queries if a specific frame rate is supported by the camera in video mode 5. (Y, 640x480). See DCAM1394_PARAM_CAP_FRAME_RATE_VID_MODE_0 for a listing of valid subparam values.

DCAM1394_PARAM_FRAME_RATE
Controls or queries the current frame rate of the camera. The subparam field is ignored. When selecting a frame rate, it should be compatible with the capability of the camera, which can be determined by querying one of the frame rate capability parameters above.

The value of this parameter may be one of the following:

```
DCAM1394_FRAME_RATE_0
DCAM1394_3_75_FPS
     The frame rate is 3.75 frames per second.

DCAM1394_FRAME_RATE_1
DCAM1394_7_5_FPS
     The frame rate is 7.5 frames per second.

DCAM1394_FRAME_RATE_2
DCAM1394_15_FPS
     The frame rate is 15 frames per second.

DCAM1394_FRAME_RATE_3
DCAM1394_30_FPS
     The frame rate is 30 frames per second.

DCAM1394_FRAME_RATE_4
```

```
DCAM1394_60_FPS
     The frame rate is 60 frames per second.
```

DCAM1394_PARAM_RING_BUFF_CAPACITY
Controls or queries the number of frames that the ring buffer may hold. This value can range between 2 and 30. The subparam field is ignored.

DCAM1394_PARAM_RING_BUFF_NUM_FRAMES_READY
Queries the number of frames in the ring buffer ready to be accessed. The subparam field is ignored.

DCAM1394_PARAM_RING_BUFF_READ_PTR_INCR
Controls or queries the number of bytes to advance the read pointer as it consumes data from the ring buffer. The subparam field is ignored.

DCAM1394_PARAM_FRAME_NUM_BYTES
Queries the number of bytes in a frame at the current video mode. The subparam field is ignored.

DCAM1394_PARAM_STATUS
Queries the parameter status. The subparam field is ignored.

The values for the parameter status is a bit field with the following values possibly set:

DCAM1394_STATUS_FRAME_RCV_DONE
Frame successfully received.

DCAM1394_STATUS_RING_BUFF_LOST_FRAME
A frame has been lost while processing the ring buffer.

DCAM1394_STATUS_PARAM_CHANGE
A parameter has been changed.

DCAM1394_STATUS_FRAME_SEQ_NUM_COUNT_OVERFLOW
Frame sequence number has reached its maximum possible value and has overflowed.

DCAM1394_STATUS_CAM_UNPLUG
Camera has been unplugged.

DCAM1394_PARAM_BRIGHTNESS
Query or control a camera feature. This feature queries or controls the brightness of the camera.

DCAM1394_SUBPARAM_PRESENCE
Indicates if the feature is available.

DCAM1394_SUBPARAM_CAP_ON_OFF
Indicates if the feature may be enabled and disabled. May only be queried.

DCAM1394_SUBPARAM_ON_OFF
Indicates if the feature is enabled.

DCAM1394_SUBPARAM_CAP_CTRL_AUTO
Indicates if the automatic control of this feature is supported by the camera. May only be queried.

DCAM1394_SUBPARAM_CAP_CTRL_MANUAL
Indicates if the manual control of this feature is supported by the camera. May only be queried.

DCAM1394_SUBPARAM_CTRL_MODE
Indicates if the feature is in auto or manual mode.

DCAM1394_SUBPARAM_MIN_VAL
Minimum value of the feature. May only be queried.

DCAM1394_SUBPARAM_MAX_VAL
Maximum value of the feature. May only be queried.

DCAM1394_SUBPARAM_VALUE
Current value of the feature.

DCAM1394_SUBPARAM_CAP_READ
Indicates if the feature may be read. May only be queried.

DCAM1394_PARAM_EXPOSURE
Query or control a camera feature. This feature queries or controls the exposure of the camera. The subparams supported by this feature are described under DCAM1394_PARAM_BRIGHTNESS.

DCAM1394_PARAM_SHARPNESS
Query or control a camera feature. This feature queries or controls the sharpness of the camera. The subparams supported by this feature are described under DCAM1394_PARAM_BRIGHTNESS.

DCAM1394_PARAM_WHITE_BALANCE
Query or control a camera feature. This feature queries or controls the white balance of the camera. The subparams supported by this feature are described under DCAM1394_PARAM_BRIGHTNESS, except for DCAM1394_SUBPARAM_VALUE. DCAM1394_SUBPARAM_VALUE is replaced by two distinct subparams.

DCAM1394_SUBPARAM_U_VALUE U or B component of the white balance.

DCAM1394_SUBPARAM_V_VALUE V or R component of the white balance.

DCAM1394_PARAM_HUE
Query or control a camera feature. This feature queries or controls the hue of the camera. The subparams supported by this feature are described under DCAM1394_PARAM_BRIGHTNESS.

DCAM1394_PARAM_SATURATION
Query or control a camera feature. This feature queries or controls the saturation of the camera. The subparams supported by this feature are described under DCAM1394_PARAM_BRIGHTNESS.

DCAM1394_PARAM_GAMMA
Query or control a camera feature. This feature queries or controls the gamma of the camera. The subparams supported by this feature are described under DCAM1394_PARAM_BRIGHTNESS.

DCAM1394_PARAM_SHUTTER
Query or control a camera feature. This feature queries or controls the sharpness of the camera. The subparams supported by this feature are described under DCAM1394_PARAM_BRIGHTNESS.

DCAM1394_PARAM_GAIN
Query or control a camera feature. This feature queries or controls the gain of the camera. The subparams supported by this feature are described under DCAM1394_PARAM_BRIGHTNESS.

DCAM1394_PARAM_IRIS
Query or control a camera feature. This feature queries or controls the iris of the camera. The subparams supported by this feature are described under DCAM1394_PARAM_BRIGHTNESS.

DCAM1394_PARAM_FOCUS
Query or control a camera feature. This feature queries or controls the focus of the camera. The subparams supported by this feature are described under DCAM1394_PARAM_BRIGHTNESS.

DCAM1394_PARAM_ZOOM
Query or control a camera feature. This feature queries or controls the zoom of the camera. The subparams supported by this feature are described under DCAM1394_PARAM_BRIGHTNESS.

DCAM1394_PARAM_PAN
Query or control a camera feature. This feature queries or controls the pan of the camera. The subparams supported by this feature are described under DCAM1394_PARAM_BRIGHTNESS.

DCAM1394_PARAM_TILT
Query or control a camera feature. This feature queries or controls the tilt of the camera.The subparams supported by this feature are described under DCAM1394_PARAM_BRIGHTNESS.

Device Special Files

`/dev/dcam`*N* Device node for isochronous input from camera.

`/dev/dcamctl`*N* Device node for camera control.

Files

`kernel/drv/sparcv9/dcam1394`	64-bit ELF kernel module.
`kernel/drv/dcam1394`	32-bit ELF kernel module.

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

See Also

`attributes(5)`, `hci1394(7D)`

1394 Trade Association Digital Camera Specification, Version 1.04 – 1996

IEEE Std 1394-2000 Standard for a High Performance Serial Bus – 2000

Name dcfs – Compression file system

Synopsis

```
#include <sys/filio.h>
#include <sys/fs/decomp.h>
```

Description The `dcfs` filesystem is a layered filesystem that you use to compress data when writing to a file and decompress upon read. The primary function of the `dcfs` filesystem is to compress individual files when constructing a boot archive and when reading or booting from the archive.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Uncommitted

See Also `boot(1M)`, `bootadm(1M)`, `fiocompress(1M)`, `attributes(5)`, `ufs(7FS)`

Notes The `dcfs` compression/decompression file system works only with UFS.

Name dda – MMS disk archiving driver

Synopsis `/devices/pseudo/dda@lun:bn`

Description The Media Management System (MMS) is a distributed removable media management system based on the *IEEE 1244* Media Management System specification. MMS provides a uniform and consistent tape interface to client applications.

The `dda` driver is a pseudo tape drive with filesystem file media. The `dda` driver is a minimal emulation of the `st`(7D) tape driver `mtio`(7I) interface with BSD no-rewind behavior and is used for MMS disk archiving. The `dda` driver should be used with a MMS MMP mount command handle. You configure DDA tapes, drives, and libraries for disk archiving using the `mmsadm`(1M) command. You also use `mmsadm`(1M) to mount and unmount `dda` media.

Configuration To set the number of disk archiving tape drives available for MMS disk archiving, edit the `/kernel/drv/dda.conf` configuration file. The default number of disk archiving tape drives is 20.

The disk archiving tape drive devlinks are located in the `/dev/dda` directory.

The DDA maximum block size is 262144 bytes and the minimum is 1 byte.

Files

`/kernel/drv/dda`	32–bit kernel module (x86)
`/kernel/drv/sparcv9/dda`	64–bit kernel module (SPARC)
`/kernel/drv/amd64/dda`	64–bit kernel module (x86)
`/kernel/drv/dda.conf`	`dda` configuration file

Attributes See `attributes`(5) for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86
Availability	SUNWmmsr
Interface Stability	Uncommitted

See Also `mt`(1), `mmsadm`(1M), `attributes`(5), `dmd`(7D), `st`(7D), `mtio`(7I)

IEEE 1244 Removable Media Standards Specification — IEEE, 2000

Name dev – Device name file system

Description The dev filesystem manages the name spaces of devices under the Solaris operating environment. The global zone's instance of the dev filesystem is mounted during boot on /dev.

A subdirectory under /dev may have unique operational semantics. Most of the common device names under /dev are created automatically by devfsadm(1M). Others, such as /dev/pts, are dynamic and reflect the operational state of the system. You can manually generate device names for newly attached hardware by invoking devfsadm(1M) or implicitly, by indirectly causing a lookup or readdir operation in the filesystem to occur. For example, you can discover a disk that was attached when the system was powered down (and generate a name for that device) by invoking format(1M)).

Files /dev Mount point for the /dev filesystem in the global zone.

See Also devfsadm(1M), format(1M), devfs(7FS)

Notes The global /dev instance cannot be unmounted.

Name devfs – Devices file system

Description The `devfs` filesystem manages a name space of all devices under the Solaris operating environment and is mounted during boot on the `/devices` name space.

The `/devices` name space is dynamic and reflects the current state of accessible devices under the Solaris operating environment. The names of all attached device instances are present under `/devices`.

The content under `/devices` is under the exclusive control of the `devfs` filesystem and cannot be changed.

The system may be configured to include a device in one of two ways:

By means of dynamic reconfiguration (DR), using, for example, `cfgadm(1M)`.

For devices driven by `driver.conf(4)` enumeration, edit the `driver.conf` file to add a new entry, then use `update_drv(1M)` to cause the system to re-read the `driver.conf` file and thereby enumerate the instance.

The device may be attached through a number of system calls and programs, including `open(2)`, `stat(2)` and `ls(1)`. During device attach, the device driver typically creates minor nodes corresponding to the device via `ddi_create_minor_node(9F)`. If the attach is successful, one or more minor nodes referring to the device are created under `/devices`.

Operations like `mknod(2)`, `mkdir(2)` and `creat(2)` are not supported in `/devices`.

Files /devices Mount point for `devfs` file system

See Also `devfsadm(1M)`, `vfstab(4)`, `attach(9E)`

Notes The `/devices` name space cannot be unmounted.

All content at or below the `/devices` name space is an implementation artifact and subject to incompatible change or removal without notification.

Name devinfo – device information driver

Description The `devinfo` driver is a private mechanism used by the `libdevinfo(3LIB)` interfaces to access kernel device configuration data and to guarantee data consistency.

Files `/devices/pseudo/devinfo@0:devinfo`

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Stability Level	Private

See Also `libdevinfo(3LIB)`, `attributes(5)`

Writing Device Drivers

Name dkio – disk control operations

Synopsis

```
#include <sys/dkio.h>
#include <sys/vtoc.h>
```

Description Disk drivers support a set of `ioctl(2)` requests for disk controller, geometry, and partition information. Basic to these `ioctl()` requests are the definitions in `<sys/dkio.h>`.

ioctls The following `ioctl()` requests set and/or retrieve the current disk controller, partitions, or geometry information on all architectures:

`DKIOCINFO`
The argument is a pointer to a `dk_cinfo` structure (described below). This structure tells the controller–type and attributes regarding bad-block processing done on the controller.

```
/*
 * Structures and definitions for disk I/O control commands
 */
#define DK_DEVLEN 16   /* device name max length, */
                       /* including unit # and NULL */
                       /* Used for controller info */
struct dk_cinfo {
     char      dki_cname[DK_DEVLEN];    /* controller name */
                                        /* (no unit #) */
     ushort_t  dki_ctype;               /* controller type */
     ushort_t  dki_flags;               /* flags */
     ushort_t  dki_cnum;                /* controller number */
     uint_t    dki_addr;                /* controller address */
     uint_t    dki_space;               /* controller bus type */
     uint_t    dki_prio;                /* interrupt priority */
     uint_t    dki_vec;                 /* interrupt vector */
     char      dki_dname[DK_DEVLEN];    /* drive name (no unit #) */
     uint_t    dki_unit;                /* unit number */
     uint_t    dki_slave;               /* slave number */
     ushort_t  dki_partition;           /* partition number */
     ushort_t  dki_maxtransfer;         /* maximum transfer size */
                                        /* in DEV_BSIZE */

     };
     /*
      * Controller types
      */

     #define DKC_UNKNOWN      0
     #define DKC_CDROM        1        /* CD-ROM, SCSI or other */
     #define DKC_WDC2880      2
     #define DKC_XXX_0        3        /* unassigned */
     #define DKC_XXX_1        4        /* unassigned */
     #define DKC_DSD5215      5
```

```
#define DKC_ACB4000      7
#define DKC_XXX_2        9         /* unassigned */
#define DKC_NCRFLOPPY    10
#define DKC_SMSFLOPPY    12
#define DKC_SCSI_CCS     13        /* SCSI CCS compatible */
#define DKC_INTEL82072   14        /* native floppy chip */
#define DKC_MD           16        /* meta-disk (virtual-disk) */
                                   /* driver */
#define DKC_INTEL82077   19        /* 82077 floppy disk */
                                   /* controller */
#define DKC_DIRECT       20        /* Intel direct attached */
                                   /* device (IDE) */
#define DKC_PCMCIA_MEM   21        /* PCMCIA memory disk-like */
                                   /* type */
#define DKC_PCMCIA_ATA   22        /* PCMCIA AT Attached type */

/*
 * Sun reserves up through 1023
 */

#define DKC_CUSTOMER_BASE  1024

/*
 * Flags
 */

#define DKI_BAD144       0x01          /* use  DEC  std  144  */
                                       /* bad  sector fwding */
#define DKI_MAPTRK       0x02          /* controller does */
                                       /* track mapping */
#define DKI_FMTTRK       0x04          /* formats only  full
                                       /* track at a time*/
#define DKI_FMTVOL       0x08          /* formats only full */
                                       /* volume at a time*/
#define DKI_FMTCYL       0x10          /* formats only full */
                                       /* cylinders at a time*/
#define DKI_HEXUNIT      0x20          /* unit number printed as */
                                       /* 3 hexdigits */
#define DKI_PCMCIA_PFD   0x40          /* PCMCIA pseudo-floppy */
                                       /* memory card */
```

DKIOCGAPART

The argument is a pointer to a `dk_allmap` structure (described below). This `ioctl()` gets the controller's notion of the current partition table for disk drive.

DKIOCSAPART

The argument is a pointer to a `dk_allmap` structure (described below). This `ioctl()` sets the controller's notion of the partition table without changing the disk itself.

```
/*
 * Partition map (part of dk_label)
 */ struct dk_map {
     daddr_t dkl_cylno;     /* starting cylinder */
     daddr_t dkl_nblk;      /* number of blocks */
     };
/*
 * Used for all partitions
 */
struct dk_allmap {
    struct dk_map    dka_map[NDKMAP];
};
```

`DKIOCGGEOM` The argument is a pointer to a `dk_geom` structure (described below). This `ioctl()` gets the controller's notion of the current geometry of the disk drive.

`DKIOCSGEOM` The argument is a pointer to a `dk_geom` structure (described below). This `ioctl()` sets the controller's notion of the geometry without changing the disk itself.

`DKIOCGVTOC` The argument is a pointer to a `vtoc` structure (described below). This `ioctl()` returns the device's current volume table of contents (VTOC.) For disks larger than 1TB, DKIOCGEXTVTOC must be used instead.

`DKIOCSVTOC` The argument is a pointer to a `vtoc` structure (described below). This `ioctl()` changes the VTOC associated with the device. For disks larger than 1TB, DKIOCSEXTVTOC must be used instead.

```
struct partition {
ushort_t      p_tag;         /* ID tag of partition */
ushort_t      p_flag;        /* permission flags */
daddr_t       p_start;       /* start sector of partition */
long          p_size;        /* # of blocks in partition */
};
```

If `DKIOCSVTOC` is used with a floppy diskette, the `p_start` field must be the first sector of a cylinder. To compute the number of sectors per cylinder, multiply the number of heads by the number of sectors per track.

```
struct vtoc {
unsigned long     v_bootinfo[3];              /* info needed by mboot
                                              /* (unsupported)*/
unsigned long     v_sanity;                   /* to verify vtoc */
                                              /* sanity */
unsigned long     v_version;                  /* layout version */
char              v_volume[LEN_DKL_VVOL];     /* volume name */
ushort_t          v_sectorsz;                 \*
     sector size in bytes*/
ushort_t          v_nparts;                   \*
```

```
     number of partitions*/
unsigned long     v_reserved[10];             /* free space */
struct partition  v_part[V_NUMPAR];           /* partition headers */
time_t            timestamp[V_NUMPAR];        /* partition timestamp */
                                              /* (unsupported) */
char              v_asciilabel[LEN_DKL_ASCII]; /* compatibility */
};

/*
* Partition permission flags
*/

#define V_UNMNT      0x01    /* Unmountable partition */
#define V_RONLY      0x10    /* Read only */

/*
* Partition identification tags
*/

#define V_UNASSIGNED   0x00  /* unassigned partition */
#define V_BOOT         0x01  /* Boot partition */
#define V_ROOT         0x02  /* Root filesystem */
#define V_SWAP         0x03  /* Swap filesystem */
#define V_USR          0x04  /* Usr filesystem */
#define V_BACKUP       0x05  /* full disk */
#define V_VAR          0x07  /* Var partition */
#define V_HOME         0x08  /* Home partition */
#define V_ALTSCTR      0x09  /* Alternate sector partition */
```

DKIOCGEXTVTOC

The argument is a pointer to an `extvtoc` structure (described below). This ioctl returns the device's current volume table of contents (VTOC). VTOC is extended to support a disk up to 2TB in size. For disks larger than 1TB this ioctl must be used instead of `DKIOCGVTOC`.

DKIOCSEXTVTOC

The argument is a pointer to an `extvtoc` structure (described below). This ioctl changes the VTOC associated with the device. VTOC is extended to support a disk up to 2TB in size. For disks larger than 1TB this ioctl must be used instead of `DKIOCSVTOC`.

```
struct extpartition {
ushort_t p_tag;         /* ID tag of partition */
ushort_t p_flag;        /* permission flags */
ushort_t p_pad[2];       /* reserved */
diskaddr_t p_start;      /* start sector no of partition */
diskaddr_t p_size;       /* # of blocks in partition */
};


struct extvtoc {
```

```
uint64_t   v_bootinfo[3]; /* info needed by mboot (unsupported) */
uint64_t   v_sanity;     /* to verify vtoc sanity */
uint64_t   v_version;    /* layout version */
char    v_volume[LEN_DKL_VVOL]; /* volume name */
ushort_t   v_sectorsz;   /* sector size in bytes */
ushort_t   v_nparts;     /* number of partitions */
ushort_t   pad[2];
uint64_t   v_reserved[10];
struct extpartition v_part[V_NUMPAR]; /* partition headers */
uint64_t timestamp[V_NUMPAR]; /* partition timestamp (unsupported)*/
char    v_asciilabel[LEN_DKL_ASCII];    /* for compatibility */
};

Partition permissions flags and identification tags
are defined the same as vtoc structure.
```

DKIOCEJECT
: If the drive supports removable media, this `ioctl()` requests the disk drive to eject its disk.

DKIOCREMOVABLE
: The argument to this `ioctl()` is an integer. After successful completion, this `ioctl()` sets that integer to a non-zero value if the drive in question has removable media. If the media is not removable, the integer is set to `0`.

DKIOCHOTPLUGGABLE
: The argument to this `ioctl()` is an integer. After successful completion, this `ioctl()` sets that integer to a non-zero value if the drive in question is hotpluggable. If the media is not hotpluggable, the integer is set to 0.

DKIOCREADONLY
: The argument to this `ioctl()` is an integer. After successful completion, this `ioctl()` sets that integer to a non-zero value if the drive in question has read-only media. If the media is writable, or not present, the integer is set to 0.

DKIOCSTATE
: This `ioctl()` blocks until the state of the drive, inserted or ejected, is changed. The argument is a pointer to a `dkio_state`, enum, whose possible enumerations are listed below. The initial value should be either the last reported state of the drive, or `DKIO_NONE`. Upon return, the enum pointed to by the argument is updated with the current state of the drive.

```
enum dkio_state {
DKIO_NONE,          /* Return disk's current state */
DKIO_EJECTED,       /* Disk state is 'ejected' */
DKIO_INSERTED       /* Disk state is 'inserted' */
};
```

DKIOCLOCK
: For devices with removable media, this `ioctl()` requests the disk drive to lock the door.

DKIOCUNLOCK
For devices with removable media, this `ioctl()` requests the disk drive to unlock the door.

DKIOCGMEDIAINFO
The argument to this `ioctl()` is a pointer to a `dk_minfo` structure. The structure indicates the type of media or the command set profile used by the drive to operate on the media. The `dk_minfo` structure also indicates the logical media block size the drive uses as the basic unit block size of operation and the raw formatted capacity of the media in number of logical blocks.

DKIOCGMEDIAINFOEXT
The argument to this `ioctl()` is a pointer to a `dk_minfo_ext` structure. The structure indicates the type of media or the command set profile used by the drive to operate on the media. The `dk_minfo_ext` structure also indicates the logical media block size the drive uses as the basic unit block size of operation, the raw formatted capacity of the media in number of logical blocks and the physical block size of the media.

```
/*
* Used for media info or profile info
*/
struct dk_minfo {
uint_t dki_media_type; /* Media type or profile info */
uint_t dki_lbsize; /* Logical blocksize of media */
diskaddr_t dki_capacity; /* Capacity as # of dki_lbsize blks */
};

/*
* Used for media info or profile info and physical blocksize
*/
struct dk_minfo_ext {
uint_t dki_media_type; /* Media type or profile info */
uint_t dki_lbsize; /* Logical blocksize of media */
diskaddr_t dki_capacity; /* Capacity as # of dki_lbsize blks */
uint_t dki_pbsize; /* Physical blocksize of media */
};


/*
* Media types or profiles known
*/
#define DK_UNKNOWN         0x00    /* Media inserted - type unknown */

/*
* SFF 8090 Specification Version 3, media types 0x01 - 0xfffe are
* retained to maintain compatibility with SFF8090.  The following
* define the optical media type.
*/
#define DK_MO_ERASABLE     0x03 /* MO Erasable */
```

```
#define DK_MO_WRITEONCE    0x04 /* MO Write once */
#define DK_AS_MO           0x05 /* AS MO */
#define DK_CDROM           0x08 /* CDROM */
#define DK_CDR             0x09 /* CD-R */
#define DK_CDRW            0x0A /* CD-RW */
#define DK_DVDROM          0x10 /* DVD-ROM */
#define DK_DVDR            0x11 /* DVD-R */
#define DK_DVDRAM          0x12 /* DVD_RAM or DVD-RW */

/*
* Media types for other rewritable magnetic media
*/
#define DK_FIXED_DISK      0x10001 /* Fixed disk SCSI or otherwise */
#define DK_FLOPPY          0x10002 /* Floppy media */
#define DK_ZIP             0x10003 /* IOMEGA ZIP media */
#define DK_JAZ             0x10004 /* IOMEGA JAZ media */
```

If the media exists and the host can obtain a current profile list, the command succeeds and returns the `dk_minfo` structure with data representing that media.

If there is no media in the drive, the command fails and the host returns an `ENXIO` error, indicating that it cannot gather the information requested.

If the profile list is not available, the host attempts to identify the media-type based on the available information.

If identification is not possible, the host returns media type `DK_UNKNOWN`. See *NOTES* for blocksize usage and capacity information.

`DKIOCSMBOOT`
The argument is a pointer to struct *mboot*.

Copies the *mboot* information supplied in the argument to the absolute sector 0 of the device. Prior to copying the information, this `ioctl()` performs the following checks on the *mboot* data:

- Ensures that the signature field is set to 0xAA55.
- Ensures that partitions do not overlap.
- On SPARC platforms, determines if the device is a removable media.

If the above verification fails, `errno` is set to `EINVAL` and the `ioctl()` command fails.

x86 Platforms — Upon successful write of *mboot*, the partition map structure maintained in the driver is updated. If the new Solaris partition is different from the previous one, the internal VTOC table maintained in the driver is set as follows:

If _SUNOS_VTOC_8 is defined:

Partition: 0. Start: 0. Capacity = Capacity of device.

Partition: 2. Start: 0. Capacity = Capacity of device.

If _SUNOS_VTOC_16 is defined:

Partition: 2. Start: 0. Size = Size specified in mboot - 2 cylinders.

Partition: 8. Start: 0. Size = Sectors/cylinder.

Partition: 9. Start: Sectors/cylinder. Size = 2 * sectors/cylinder

To determine if the Solaris partition has changed:

If either offset or the size of the Solaris partition is different from the previous one then it shall be deemed to have changed. In all other cases, the internal VTOC info remains as before.

SPARC Platforms — The VTOC label and *mboot* both occupy the same location, namely sector 0. As a result, following the successful write of *mboot* info, the internal VTOC table maintained in the driver is set as follows:

Partition: 0. Start: 0. Size = Capacity of device.

Partition: 2. Start: 0. Size = Capacity of device.

See the NOTES section for usage of DKIOCSMBOOT when modifying Solaris partitions.

`DKIOCGETVOLCAP`

This ioctl provides information and status of available capabilities.

vc_info is a bitmap and the valid flag values are:

```
DKV_ABR_CAP -  Capable of application-based recovery
DKV_DMR_CAP -  Ability to read specific copy of data when
               multiple copies exist. For example, in a two
               way mirror, this ioctl is used to read each
               side of the mirror.
```

vc_set is a bitmap and the valid flag values are:

```
DKV_ABR_CAP - This flag is set if ABR has been set on a device
              that supports ABR functionality.
DKV_DMR_CAP - Directed read has been enabled.
```

These capabilities are not required to be persistent across a system reboot and their persistence depends upon the implementation. For example, if the ABR capability for a DRL mirror simply clears the dirty-region list and subsequently stops updating this list, there is no reason for persistence because the VM recovery is a no-op. Conversely, if the ABR capability is applied to a non-DRL mirror to indicate that the VM should not perform a full recovery of the mirror following a system crash, the capability must be persistent so that the VM know whether or not to perform recovery.

Return Errors:

`EINVAL` Invalid device for this operation.

`ENOTSUP` Functionality that is attempted to be set is not supported.

`DKIOCSETVOLCAP`

This ioctl sets the available capabilities for the device. If a capability flag is not set in vc_set, that capability is cleared.

vc_info flags are ignored

vc_set valid flags are:

```
DKV_ABR_CAP - Flag to set application-based recovery. A device can
              successfully support ABR only if it is capable.
DKV_DMR_CAP - Flag to set directed read.


int
ioctl(int , DKIODMR, vol_directed_rd *);
```

`DKIODMR`

This ioctl allows highly available applications to perform round-robin reads from the underlying devices of a replicated device.

```
vdr_offset      - offset at which the read should occur.
vdr_nbytes      - number of bytes to be read
vdr_bytesread   - number of bytes successfully read by the kernel.
vdr_data        - pointer to a user allocated buffer to return the
                  data read
vdr_side        - side to be read. Initialized to DKV_SIDE_INIT
vdr_side_name   - The volume name that has been read.

Valid vdr_flags are:
      DKV_DMR_NEXT_SIDE (set by user)
      DKV_DMR_DONE (return value)
      DKV_DMR_ERROR (return value)
      DKV_DMR_SUCCESS(return value)
      DKV_DMR_SHORT(return value)
```

The calling sequence is as follows: The caller sets the *vdr_flags* to `DK_DMR_NEXT_SIDE` and *vdr_side* to `DKV_SIDE_INIT` at the start. Subsequent calls should be made without any changes to these values. If they are changed the results of the ioctl are indeterminate.

When `DKV_SIDE_INIT` is set, the call results in the kernel reading from the first side. The kernel updates *vdr_side* to indicate the side that was read, and *vdr_side_name* to contain the name of that side. *vdr_data* contains the data that was read. Therefore to perform a round-robin read all of the valid sides, there is no need for the caller to change the contents of *vdr_side*.

Subsequent ioctl calls result in reads from the next valid side until all valid sides have been read. On success, the kernel sets `DKV_DMR_SUCCESS`. The following table shows the values of *vdr_flags* that are returned when an error occurs:

```
vdr_flags    |   vdr_side        |       Notes
-------------|-------------------|---------------------------
DKV_DMR_ERROR|   DKV_SIDE_INIT   |   No valid side to read
DKV_DMR_DONE |   Not Init side   |   All valid sides read
DKV_DMR_SHORT|   Any value       |   Bytes requested cannot
                                     be read. vdr_bytesread
                                     set to bytes actually
                                     read.
```

Typical code fragment:

```
enable->vc_set |= DKV_ABR_SET;
retval = ioctl(filedes, DKIOSETVOLCAP, enable);
if (retval != EINVAL || retval != ENOTSUP) {
        if (info->vc_set & DKV_DMR_SET) {
                dr->vdr_flags |= DKV_DMR_NEXT_SIDE;
                dr->vdr_side = DKV_SIDE_INIT;
                dr->vdr_nbytes = 1024;
                dr->vdr_offset = 0xff00;
                do {
                        rval =ioctl(fildes, DKIODMR, dr);
                        if (rval != EINVAL) {
                                /* Process data */
                        }
                } while (rval != EINVAL || dr->vdr_flags &
                    (DKV_DMR_DONE | DKV_DMR_ERROR | DKV_DMR_SHORT)
        }
}
```

RETURN VALUES Upon successful completion, the value returned is `0`. Otherwise, `-1` is returned and `errno` is set to indicate the error.

x86 Only The following `ioctl()` requests set and/or retrieve the current disk controller, partitions, or geometry information on the x86 architecture.

`DKIOCG_PHYGEOM`
The argument is a pointer to a `dk_geom` structure (described below). This `ioctl()` gets the driver's notion of the physical geometry of the disk drive. It is functionally identical to the `DKIOCGGEOM` `ioctl()`.

`DKIOCG_VIRTGEOM`
The argument is a pointer to a `dk_geom` structure (described below). This `ioctl()` gets the controller's (and hence the driver's) notion of the virtual geometry of the disk drive. Virtual geometry is a view of the disk geometry maintained by the firmware in a host bus adapter or disk controller. If the disk is larger than 8 Gbytes, this ioctl fails because a CHS-based

geometry is not relevant or useful for this drive.

```
/*
* Definition of a disk's geometry
*/
*/struct dk_geom {
unsigned shor    dkg_ncyl;             /* # of data cylinders */
unsigned shor    dkg_acyl;             /* # of alternate cylinders */
unsigned short   dkg_bcyl;             /* cyl offset (for fixed head */
                                       /* area) */
unsigned short   dkg_nhead;            /* # of heads */
unsigned short   dkg_obs1;             /* obsolete */
unsigned short   dkg_nsect;            /* # of sectors per track*/
unsigned short   dkg_intrlv;           /* interleave factor */
unsigned short   dkg_obs2;             /* obsolete */
unsigned short   dkg_obs3;             /* obsolete */
unsigned short   dkg_apc;              /* alternates per cylinder */
                                       /* (SCSI only) */
unsigned short   dkg_rpm;              /* revolutions per min*/
unsigned short   dkg_pcyl;             /* # of physical cylinders */
unsigned short   dkg_write_reinstruct; /* # sectors to skip, writes*/
unsigned short   dkg_read_reinstruct;  /* # sectors to skip, reads*/
unsigned short   dkg_extra[7];         /* for compatible expansion*/
};
```

DKIOCADDBAD
: This `ioctl()` forces the driver to re-examine the alternates slice and rebuild the internal bad block map accordingly. It should be used whenever the alternates slice is changed by any method other than the addbadsec(1M) or format(1M) utilities. `DKIOCADDBAD` can only be used for software remapping on IDE drives; SCSI drives use hardware remapping of alternate sectors.

DKIOCPARTINFO
: The argument is a pointer to a `part_info` structure (described below). This `ioctl()` gets the driver's notion of the size and extent of the partition or slice indicated by the file descriptor argument.

```
/*
 * Used by applications to get partition or slice information
 */
struct part_info {
daddr_t    p_start;
int        p_length;
      };
```

DKIOCEXTPARTINFO
: The argument is a pointer to an `extpart_info` structure (described below). This ioctl gets the driver's notion of the size and extent of the partition or slice indicated by the file descriptor argument. On disks larger than 1TB, this ioctl must be used instead of `DKIOCPARTINFO`.

```
/*
* Used by applications to get partition or slice information
*/
struct extpart_info {
diskkaddr_t      p_start;
diskaddr_t       p_length;
};
```

DKIOCSETEXTPART

This ioctl is used to update the in-memory copy of the logical drive information maintained by the driver. The ioctl takes no arguments. It causes a re-read of the partition information and recreation of minor nodes if required. Prior to updating the data structures, the ioctl ensures that the partitions do not overlap. Device nodes are created only for valid partition entries. If there is any change in the partition offset, size or ID from the previous read, the partition is deemed to have been changed and hence the device nodes are recreated. Any modification to any of the logical partitions results in the recreation of all logical device nodes.

See Also addbadsec(1M), fdisk(1M), format(1M), ioctl(2), cdio(7I), cmdk(7D), fdio(7I), hdio(7I), sd(7D)

Notes Blocksize information provided in DKIOCGMEDIAINFO is the size (in bytes) of the device's basic unit of operation and can differ from the blocksize that the Solaris operating environment exports to the user. Capacity information provided in the DKIOCGMEDIAINFO are for reference only and you are advised to use the values returned by DKIOCGGEOM or other appropriate ioctl for accessing data using the standard interfaces.

For x86 only: If the DKIOCSMBOOT command is used to modify the Solaris partitions, the VTOC information should also be set appropriately to reflect the changes to partition. Failure to do so leads to unexpected results when the device is closed and reopened fresh at a later time. This is because a default VTOC is assumed by driver when a Solaris partition is changed. The default VTOC persists until the ioctl DKIOCSVTOC is called to modify VTOC or the device is closed and reopened. At that point, the old valid VTOC is read from the disk if it is still available.

Name dlcosmk – Data Layer Class of Service Marker

Description The `dlcosmk` marker is an action module that is executed as a result of classifying or metering packets. It marks the packet with a user priority defined by the *IEEE 801.D* standard. This feature is only possible on a VLAN device.

The 3-bit user priority is part of the *802.1Q* VLAN header tag that is part of the ethernet header (carrying the IP packet).

Statistics The `dlcosmk` module exports the following statistics through `kstat`:

Global statistics:

```
module: dlcosmk                         instance: <action id>
 name:  dlcosmk statistics              class <action name>
        crtime
        snaptime
        b_band                          <b_band value>
        dl_max                          <dl_max value>
        usr_pri                         <configured CoS>
        npackets                        <number of packets>
        epackets                        <number of packets in error>
        ipackets                        <number of packets not processed>
```

Files `/kernel/ipp/sparcv9/dlcosmk` 64-bit module (SPARC only.)

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWqos

See Also `ipqosconf(1M)`, `dscpmk(7ipp)`, `flowacct(7ipp)`, `ipqos(7ipp)`, `ipgpc(7ipp)`, `tokenmt(7ipp)`, `tswtclmt(7ipp)`

Name dlpi – Data Link Provider Interface

Synopsis

```
#include <sys/dlpi.h>
```

Description SunOS STREAMS-based device drivers wishing to support the STREAMS TCP/IP and other STREAMS-based networking protocol suite implementations support Version 2 of the Data Link Provider Interface (DLPI). DLPI V2 enables a data link service user to access and use any of a variety of conforming data link service providers without special knowledge of the provider's protocol. Specifically, the interface is intended to support Ethernet, X.25 LAPB, SDLC, ISDN LAPD, CSMA/CD, FDDI, token ring, token bus, Bisync, and other datalink-level protocols.

The interface specifies access to the data link service provider in the form of `M_PROTO` and `M_PCPROTO` type STREAMS messages and does not define a specific protocol implementation. The interface defines the syntax and semantics of primitives exchanged between the data link user and the data link provider to attach a physical device with physical-level address to a stream, bind a datalink-level address to the stream, get implementation-specific information from the data link provider, exchange data with a peer data link user in one of three communication modes (connection, connectionless, acknowledged connectionless), enable/disable multicast group and promiscuous mode reception of datalink frames, get and set the physical address associated with a stream, and several other operations.

Solaris conforms to The Open Group Technical Standard for *DLPI, Version 2*. For free access to this specification, point your browser to *www.opengroup.org/pubs/catalog/c811.htm*. Solaris also provides extensions to the DLPI standard, as detailed in this man page.

Solaris-SPECIFIC Dlpi Extensions

Notification Support

Enables DLPI consumers to register for notification when events of interest occur at the DLPI provider. The negotiation can be performed on any attached DLPI stream, and begins with the DLPI consumer, sending a DL_NOTIFY_REQ to the provider, which is an M_PROTO message with the following payload:

```
typedef struct {
        t_uscalar_t     dl_primitive;
        uint32_t        dl_notifications;
        uint32_t        dl_timelimit;
} dl_notify_req_t;
```

The dl_primitive field must be set to DL_NOTIFY_REQ; the dl_timelimit field is reserved for future use and must be set to zero. The dl_notifications field is a bitmask containing the event types the consumer is interested in receiving, and must be zero or more of:

```
DL_NOTE_LINK_DOWN         Notify when link has gone down
DL_NOTE_LINK_UP           Notify when link has come up
DL_NOTE_PHYS_ADDR         Notify when address changes
DL_NOTE_SDU_SIZE          Notify when MTU changes
DL_NOTE_SPEED             Notify when speed changes
DL_NOTE_PROMISC_ON_PHYS   Notify when DL_PROMISC_PHYS is set
DL_NOTE_PROMISC_OFF_PHYS  Notify when DL_PROMISC_PHYS is cleared
```

Consumers might find it useful to send a DL_NOTIFY_REQ message with no requested types to check if the DLPI provider supports the extension.

Upon receiving the DL_NOTIFY_REQ, the DLPI provider must generate a DL_NOTIFY_ACK, which is an M_PROTO message with the following payload:

```
typedef struct {
        t_uscalar_t     dl_primitive;
        uint32_t        dl_notifications;
} dl_notify_ack_t;
```

The `dl_primitive` field must be set to DL_NOTIFY_ACK. The `dl_notifications` field must include any notifications that the provider supports, along with any other unrequested notifications that the provider supports. However, regardless of the notifications the provider supports, it is restricted to sending only DL_NOTIFY_IND messages (see below) that were requested in the DL_NOTIFY_REQ.

Since there are additional notification types which are not yet available for public use, DLPI consumers and providers must take care when inspecting and setting the `dl_notifications` field. Specifically, consumers must be careful to only request the above notification types, and providers must be careful to not include any unrecognized notification types in the `dl_notifications` field when constructing the DL_NOTIFY_ACK. In addition, DL_NOTIFY_IND's that are received with undocumented dl_notification or dl_data values must be ignored.

DLPI consumers might receive a DL_ERROR_ACK message (with dl_error_primitive set to DL_NOTIFY_REQ) in response to the initial DL_NOTIFY_REQ message. This message indicates that the DLPI provider does not support the DLPI notification extension. Otherwise, the DLPI consumer receives a DL_NOTIFY_ACK and should expect to receive DL_NOTIFY_IND messages for any types that it requested that were still set in it. The DL_NOTIFY_IND is an M_PROTO message with the following payload:

```
typedef struct {
        t_uscalar_t     dl_primitive;
        uint32_t        dl_notification;
        uint32_t        dl_data;
        t_uscalar_t     dl_addr_length;
        t_uscalar_t     dl_addr_offset;
} dl_notify_ind_t;
```

The `dl_primitive` field must be set to DL_NOTIFY_IND, and the `dl_notification` field must be set to the event type that has occurred (for example, DL_NOTE_LINK_DOWN). Only a single event type can be set in each DL_NOTIFY_IND.

For the DL_NOTE_SPEED event type, `dl_data` must be set to the current interface speed in kilobits per second. For the DL_NOTE_PHYS_ADDR event type, dl_data must be set to DL_CURR_PHYS_ADDR. For the DL_NOTE_SDU_SIZE event type, `dl_data` must be set to the current MTU in bytes. Otherwise, `dl_data` must be set to zero.

For the DL_NOTE_PHYS_ADDR event type, the `dl_addr_length` field must be set to the length of the address, and the `dl_addr_offset` field must be set to offset of the first byte of the address, relative to b_rptr (for example, if the address immediately follows the `dl_notify_ind` structure, `dl_addr_offset` is set to 'sizeof (`dl_notify_ind`)'). For all other event types, the `dl_addr_length` and `dl_addr_offset` fields

must be set to zero by DLPI providers and ignored by DLPI consumers.

In addition to generating DL_NOTIFY_IND messages when a requested event has occurred, the DLPI provider must initially generate one or more DL_NOTIFY_IND messages to notify the DLPI consumer of the the current state of the interface. For instance, if the consumer has requested DL_NOTE_LINK_UP | DL_NOTE_LINK_DOWN, the provider must send a DL_NOTIFY_IND containing the current state of the link (either DL_NOTE_LINK_UP or DL_NOTE_LINK_DOWN) after sending the DL_NOTIFY_ACK.

For the initial DL_NOTIFY_IND message, the DLPI provider is strongly recommended against sending DL_NOTE_LINK_DOWN, even if the interface is still initializing and is not yet ready to send or receive packets. Instead, either delaying the DL_NOTIFY_IND message until the interface is ready or optimistically reporting DL_NOTIFY_LINK_UP and subsequently reporting DL_NOTE_LINK_DOWN if the negotation fails is strongly preferred. This prevents DL_NOTIFY_IND consumers from needlessly triggering network failover operations and logging error messages during network interface initialization.

The DLPI provider must continue to generate DL_NOTIFY_IND messages until it receives a new DL_NOTIFY_REQ message or the DLPI stream is detached (or closed). Further, a DLPI style 2 provider must keep track of the requested events after a DL_DETACH_REQ operation, and if a subsequent DL_ATTACH_REQ is received, it must send gratuitous DL_NOTIFY_IND messages to notify the consumer of the current state of the device, since the state might have changed while detached (or the consumer might have simply discarded its previous state).

Passive Consumers of Aggregated Links

Solaris link aggregations as configured by dladm(1M) export DLPI nodes for both the link aggregation, and individual links that comprises the aggregation, to allow observability of the aggregated links. To allow applications such as snoop(1M) to open those individual aggregated links while disallowing other consumers such as ip(7P), DL_PASSIVE_REQ (a DLPI primitive), must be issued by snoop(1M) and similar applications.

The DL_PASSIVE_REQ primitive is an M_PROTO message containing the following payload:

```
typedef struct {
        t_uscalar_t     dl_primitive;
} dl_passive_req_t;
```

Issuing this primitive allows the consumer of a DLPI link to coexist with a link aggregation that also uses the link. Such a consumer is considered `passive`.

Consumers that don't use this primitive while an aggregation is using the link receive DL_SYSERR/EBUSY when issuing the following DLPI primitives:

```
DL_BIND_REQ
DL_ENABMULTI_REQ
DL_PROMISCON_REQ
DL_AGGR_REQ
DL_UNAGGR_REQ
DL_CONTROL_REQ
DL_SET_PHYS_ADDR_REQ
```

A consumer that has not issued a DL_PASSIVE_REQ and has successfully issued one of the above primitives is considered `active`.

The creation of a link aggregation using dladm(1M) fails if one of the links included in the aggregation has an `active` consumer, but succeeds if the links do not have any DLPI consumers or only `passive` consumers.

Raw Mode

The `DLIOCRAW` ioctl function is used by some DLPI applications, most notably the snoop(1M) command. The `DLIOCRAW` command puts the stream into a raw mode, which, upon receive, causes the the full MAC-level packet to be sent upstream in an `M_DATA` message instead of it being transformed into the `DL_UNITDATA_IND` form normally used for reporting incoming packets. Packet SAP filtering is still performed on streams that are in raw mode. If a stream user wants to receive all incoming packets it must also select the appropriate promiscuous modes. After successfully selecting raw mode, the application is also allowed to send fully formatted packets to the provider as `M_DATA` messages for transmission. `DLIOCRAW` takes no arguments. Once enabled, the stream remains in this mode until closed.

Native Mode

Some DLPI providers are able to represent their link layer using more than one link-layer format. In this case, the default link-layer format can minimize impact to applications, but might not allow truly `native` link-layer headers to be sent or received. DLPI consumers who wish to use the native link-layer format can use DLIOCNATIVE to transition the stream. DLIOCNATIVE takes no arguments and returns the DLPI mac type associated with the new link-layer format upon success. Once enabled, the stream remains in this mode until closed. Note that DLIOCNATIVE does not enable transition between dissimilar DLPI mac types and (aside from the link-layer format), the new DLPI mac type is guaranteed to be semantically identical. In particular, the SAP space and addressing format are not affected and the effect of DLIOCNATIVE is only visible when in raw mode, though any subsequent DL_INFO_REQ requests generate responses with dl_mac_type set to the native DLPI type.

Margin

While a DLPI provider provides its maximum SDU via dl_max_sdu in DL_INFO_ACK messages, this value typically represents a

standard maximum SDU for the provider's media (1500 for Ethernet for example), and not necessarily the absolute maximum amount of data that the provider is able to transmit in a given data unit. The `margin` "is the extra amount of data in bytes that the provider can transmit beyond its advertised maximum SDU. For example, if a DL_ETHER provider can handle packets whose payload section is no greater than 1522 bytes and its dl_max_sdu is set to 1500 (as is typical for Ethernet), then the margin would be 22. If a provider supports a non-zero margin, it implements the DLIOCMARGININFO ioctl, whose data is a t_uscalar_t representing the margin size.

DL_ETHER-SPECIFIC Dlpi Semantics

VLAN Support

Traditional VLAN Access

Some `DL_ETHER` `DLPI` providers support *IEEE 802.1Q* Virtual LANs (VLAN). For these providers, traffic for a particular VLAN can be accessed by opening a VLAN data-link.

Unless raw mode is enabled, a DLPI stream bound to a VLAN data-link behaves no differently than a traditional DLPI stream. As with non-VLAN data-link access, data must be sent to a DLPI provider without link-layer headers (which are added by the provider) and received data is passed to interested DLPI consumers without link-layer headers. As a result, DLPI consumers not require special-case logic to implement VLAN access.

SAP-Based VLAN Access

As per *IEEE 802.1Q*, all VLAN traffic is sent using Ether- Type 0x8100, meaning that in addition to directly opening a VLAN data-link, all VLAN traffic for a given underline data-link can also be accessed by opening the underlying data-link and binding to SAP 0x8100. Accordingly, all VLAN traffic (regardless of VLAN ID) can be sent and received by the DLPI consumer. However, even when raw mode is disabled, packets are received starting with their VLAN headers and must be sent to the DLPI provider with their VLAN headers already pre-pended (but without Ethernet headers). Because adhering to these semantics requires each DLPI consumer to have specialized knowledge of VLANs, VLANs should only be accessed in this way when the traditional VLAN access method is insufficient (for example, because access to all VLAN traffic, regardless of VLAN ID, is needed).

Because all VLAN traffic is sent with SAP 0x8100, VLAN traffic not filtered at the physical (`DL_PROMISC_PHYS`) level is also visible if a DLPI consumer enables promiscuous mode of a

stream at the DL_PROMISC_SAP level. As mentioned earlier, these packets are received starting with their VLAN headers if raw mode is not enabled.

QoS Support

The *IEEE 802.1Q* standard defines eight classes of priority values used by QoS traffic control of Ethernet packets. Although the priority values are encoded in the *802.1Q* tags, they can be used independently from VLANs. In particular, a special priority tagged packet (with VLAN ID zero but priority bits non-zero) does not belong to any VLAN.

The priority value can be set on either a per-stream or per-packet basis. DLPI consumers can specify the per-stream priority using the DL_UDQOS_REQ request (the priority value remains unchanged until the next DL_UDQOS_REQ) and also specify the per-packet priority value using the b_band field of a M_DATA message or the dl_priority field of a DL_UNITDATA_REQ.

Raw Mode

SAP-Based VLAN Access

When raw mode is enabled, the complete, unmodified MAC- level packet (including Ethernet and VLAN headers) is passed to interested DLPI consumers. Similarly, the entire MAC-level packet (including Ethernet and VLAN headers) must be sent to the DLPI provider for transmission. Note that the priority value specified in the b_band field can be overridden by encoding the priority value (if any) into the VLAN header.

Traditional VLAN Access

When raw mode is enabled, only packets with the correct VLAN ID are passed up to interested DLPI consumers. With the exception of priority-tagged packets, DLPI providers must strip off the VLAN headers (while retaining the preceding Ethernet headers) before sending up the packets. For priority-tagged packets, DLPI providers must use the reserved tag 0 to encode the VLAN TCI and send up the packets.

On the transmit-side, DLPI consumers must send the packets down to the DLPI providers without the VLAN headers (but with the Ethernet headers) unless certain QoS support is required. If QoS support is needed, the packet can have the VLAN header to indicate the priority value, however its VLAN ID must be zero. The DLPI providers then insert the VLAN tags or encode the VLAN tags using the priority value specified in the VLAN headers and send the packets.

Files Files in or under /dev.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability (Notification support/Passive mode behavior)	Committed

See Also dladm(1M), snoop(1M), libdlpi(3LIB), gld(7D), ip(7P)

Notes A Solaris DLPI link name consists of a *DLPI provider name* followed by a numeric *PPA* (physical point of attachment).

The DLPI provider name must be between 1 and 16 characters in length, though names between 3 and 8 characters are preferred. The DLPI provider name can consist of any alphanumeric character (a-z, A-Z, 0-9), and the underscore (_). The first and last character of the DLPI provider name cannot be a digit.

The PPA must be a number between `0` and `4294967294` inclusive. Leading zeroes are not permitted.

Name dm2s – loadable STREAMS driver

Synopsis `dm2s@0`

Description The `dm2s` module is a loadable STREAMS driver that provides synchronous serial communication support for DSCP communication. `dm2s` is specific to the SPARC Enterprise Server family.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWdscpr
Interface Stability	Private

See Also `attributes(5)`

Name dmd – Driver for MMS drive managers

Synopsis

```
/devices/pseudo/dmd@0:watcher

/devices/pseudo/dmd@0:ndrm

/devices/pseudo/dmd@0:stat
```

Description The Media Management System (MMS) is a distributed removable media management system based on the *IEEE 1244* Media Management System specification. MMS provides a uniform, consistent tape interface to client applications.

Drives managed by MMS have a drive manager on each host that processes tapes and accesses the drive. The drive manager provides device-specific operations and label processing. (Note: only ANSI labels are currently supported.) The handle returned by the MMS MMP mount command belongs to the drive manager driver (`dmd`). When `dmd` receives an open or close system call, it passes it to the drive manager for processing. The drive manager then validates the file according to the file disposition flags. Note that only O_RDONLY, O_WRONLY and O_RDWR flags are used in validation. If O_APPEND is specified, the drive manager positions the file as if append is specified in the mount command and the record format and blocksize are set.

ioctls The Device Manager driver supports the MTIOCTOP, MTREW, MTRETEN, MTFSF, MTFSR, MTBSF, MTBSR, MTWEOF, MTEOM, MTSRSZ, MTGRSZ, MTIOCGET and MTIOCLRERR driver ioctls. In addition, MMS provides the ioctls shown below for the device manager driver. (Note that MMS_LOCATE is supported in both mms and raw modes and MMS_LOCATE is supported only in mms mode).

```
MMS_BLK_LIMIT - returns the block limit of a drive.
                arg:

        typedef struct  mms_blk_limit {
                uint64_t        mms_max;        /* Max blocksize */
                uint32_t        mms_min;        /* Min blocksize */
                uint32_t        mms_gran;       /* granularity */
        }       mms_blk_limit_t;

  MMS_GET_POS - gets the current position.
                arg:

        typedef struct  mms_pos {
                uint32_t        mms_type;       /* LBN or non LBN) */
                uint64_t        mms_pos;
        }       mms_pos_t;

        /* Value of mms_pos_type */
        #define MMS_LBYTEN              1       /* logical byte */
        #define MMS_LBLKN               2       /* logical block */
```

```
  MMS_LOCATE -  locate to a specific location.
                arg: arg must be a mms_pos_t obtained from a
                previous MMS_GET_POS.

  MMS_GET_CAPACITY - returns the tape's capacity.
                     arg:
        typedef struct mms_capacity {
                /*
                 * Capacity is in megabytes (1048576)
                 */
                uint64_t        mms_max;        /* cartridge capacity */
                uint64_t        mms_avail;      /* amount available */
                                                        /* from EOD */
                uint32_t        mms_pc_avail;   /* percent available */
        }       mms_capacity_t;

  MMS_SET_DENSITY - sets the tape's density.
                    arg:
        typedef struct  mms_density {
                uint32_t        mms_den;
        }       mms_density_t;

  MMS_GET_DENSITY - gets the current tape density.
                    arg:
        typedef struct  mms_density {
                uint32_t        mms_den;
        }       mms_density_t;

  MMS_INQUIRY - gets inquiry data of drive.
                arg:
        typedef struct  mms_inquiry     {
                uchar_t         mms_inq[64];
        }       mms_inquiry_t;


Status of dmd

MMS uses the MTIOCGET ioctl to report status.
The status code in mt_dsreg is defined as follows:

        /usr/include/sys/scsi/generic/status.h

        #define STATUS_MASK                     0x3E
        #define STATUS_GOOD                     0x00
        #define STATUS_CHECK                    0x02
        #define STATUS_MET                      0x04
        #define STATUS_BUSY                     0x08
        #define STATUS_INTERMEDIATE             0x10
```

```
        #define STATUS_SCSI2                    0x20
        #define STATUS_INTERMEDIATE_MET         0x14
        #define STATUS_RESERVATION_CONFLICT     0x18
        #define STATUS_TERMINATED               0x22
        #define STATUS_QFULL                    0x28
        #define STATUS_ACA_ACTIVE               0x30

The error code in mt_erreg are defined
as follows:

        /usr/include/sys/scsi/generic/sense.h

        /*
         * Sense Key values for Extended Sense.
         */

        #define KEY_NO_SENSE            0x00
        #define KEY_RECOVERABLE_ERROR   0x01
        #define KEY_NOT_READY           0x02
        #define KEY_MEDIUM_ERROR        0x03
        #define KEY_HARDWARE_ERROR      0x04
        #define KEY_ILLEGAL_REQUEST     0x05
        #define KEY_UNIT_ATTENTION      0x06
        #define KEY_WRITE_PROTECT       0x07
        #define KEY_DATA_PROTECT        KEY_WRITE_PROTECT
        #define KEY_BLANK_CHECK         0x08
        #define KEY_VENDOR_UNIQUE       0x09
        #define KEY_COPY_ABORTED        0x0A
        #define KEY_ABORTED_COMMAND     0x0B
        #define KEY_EQUAL               0x0C
        #define KEY_VOLUME_OVERFLOW     0x0D
        #define KEY_MISCOMPARE          0x0E
        #define KEY_RESERVED            0x0F


 /usr/include/sys/scsi/impl/sense.h:
        /* Status returned by driver */
        #define SUN_KEY_FATAL           0x10    /* handshake failure */
        #define SUN_KEY_TIMEOUT         0x11    /* command timeout */
        #define SUN_KEY_EOF             0x12    /* eof hit */
        #define SUN_KEY_EOT             0x13    /* eot hit */
        #define SUN_KEY_LENGTH          0x14    /* length error */
        #define SUN_KEY_BOT             0x15    /* bot hit */
        #define SUN_KEY_WRONGMEDIA      0x16    /* wrong tape media */
        #define NUM_IMPL_SENSE_KEYS     7       /* seven extra keys */


In MMS mode, the following values may be in mt_erreg when using
```

```
mms mode. They are defined in mms.h

        #define MMS_KEY_BOF     0xe0    /* beginning of file. */
        #define MMS_KEY_EOF     0xe1    /* end of file. */
```

Files

`/kernel/drv/dmd`	32–bit kernel module (x86)
`/kernel/drv/sparcv9/dmd`	64–bit kernel module (SPARC)
`/kernel/drv/amd64/dmd`	64–bit kernel module (x86)
`/kernel/drv/dmd.conf`	Configuration file

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86
Availability	SUNWmmsr
Interface Stability	Uncommitted

See Also `mt(1)`, `mmsadm(1M)`, `attributes(5)`, `dda(7D)`, `st(7D)`, `mtio(7I)`

IEEE 1244 Removable Media Standards Specification — IEEE, 2000

Name dmfe – Davicom Fast Ethernet driver for Davicom DM9102A

Synopsis `/kernel/drv/sparcv9/dmfe`

Description The `dmfe` Ethernet device provides 100Base-TX networking interfaces using the Davicom DM9102A chip, which incorporates its own internal transceiver.

The `dmfe` driver functions include controller initialization, frame transmit and receive, promiscuous and multicast support, and error recovery and reporting. Multiple controllers installed within the system are supported by the driver.

The 100Base-TX standard specifies an auto-negotiation protocol to automatically select the mode and speed of operation. The internal transceiver is capable of performing auto-negotiation with the remote-end of the link (link partner) and receives the capabilities of the remote end. It selects the highest common denominator mode of operation based on the priorities. The internal transceiver also supports a forced-mode of operation under which the driver selects the operational mode.

Application Programming Interface The `/dev/dmfe` cloning character-special device is used to access all Davicom DM9102A devices installed in the system.

You must send an explicit DL_ATTACH_REQ message to associate the opened stream with a particular device (ppa). The ppa ID is interpreted as an unsigned integer data type and indicates the corresponding device instance (unit) number. If the ppa field value does not correspond to a valid device instance number for this system, an error (DL_ERROR_ACK) is returned. The device is initialized on first attach and de-initialized (stopped) at last detach.

The values returned by the driver in the DL_INFO_ACK primitive in response to a DL_INFO_REQ are as follows:

- Maximum SDU is 1500 (ETHERMTU - defined in `sys/ethernet.h`).
- Minimum SDU is 0.
- DLSAP address length is 8.
- MAC type is DL_ETHER.
- The sap length value is -2, meaning the physical address component is followed immediately by a 2-byte sap component within the DLSAP address.
- The broadcast address value is the Ethernet/IEEE broadcast address (FF:FF:FF:FF:FF:FF).

Once in the DL_ATTACHED state, you must send a DL_BIND_REQ to associate a particular Service Access Point (SAP) with the stream.

Configuration By default, the `dmfe` driver performs auto-negotiation to select the speed and mode of the link. Link speed and mode can be 100 Mbps (full or half-duplex) or 10 Mbps (full or half-duplex) as described in the 100Base-TX standard.

The auto-negotiation protocol automatically selects speed mode (either 100 Mbps or 10 Mbps) and operation mode (either full-duplex or half-duplex) as the highest common denominator supported by both link partners. Because the `dmfe` device supports all modes, this effectively selects the highest-throughput mode supported by the other device.

Alternatively, you can explicitly specify the link parameters by adding entries to the `dmfe` driver configuration file (`/kernel/drv/dmfe.conf`). You can set the `speed` parameter to 10 or 100 to force `dmfe` devices to operate at the specified speed. Additionally, you can set the full-duplex parameter to 0 or 1 to disable or force full-duplex operation, respectively.

Note that specifying either "speed" or "full-duplex" explicitly disables auto-negotiation. To enable the driver to determine the appropriate setting for each parameter, you should always set both parameters. If it is necessary to force either speed or duplex setting (for example, because the `dmfe` device is connected to an ancient device or hub that does not support auto-negotiation), both parameters should be explicitly specified to match the requirements of the external device.

Files

`/dev/dmfe`	Character special device
`/kernel/drv/dmfe`	32–bit kernel module (x86)
`/kernel/drv/sparcv9/dmfe`	64–bit kernel module (SPARC)
`/kernel/drv/amd64/dmfe`	64–bit kernel module (x86)
`/kernel/drv/dmfe.conf`	`dmfe` configuration file

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86

See Also

`attributes(5)`, `gld(7D)`, `dlpi(7P)`, `streamio(7I)`

Writing Device Drivers

STREAMS Programming Guide

Network Interfaces Programmer's Guide

Name dnet – Ethernet driver for DEC 21040, 21041, 21140 Ethernet cards

Synopsis `/kernel/drv/dnet`

Description The `dnet` Ethernet driver is a multithreaded, loadable, clonable, STREAMS GLD driver. Multiple controllers installed within the system are supported by the driver. The `dnet` driver functions include controller initialization, frame transmit and receive, functional addresses, promiscuous and multicast support, and error recovery and reporting.

Application Programming Interface The cloning character-special device, `/dev/dnet`, is used to access all DEC 21040/21041/21140 devices installed in the system.

The `dnet` driver is dependent on `/kernel/misc/gld`, a loadable kernel module that provides the `dnet` driver with the `DLPI` and `STREAMS` functionality required of a LAN driver. See `gld(7D)` for more details on the primitives supported by the driver.

The device is initialized on the first attach and de-initialized (stopped) on the last detach.

The values returned by the driver in the `DL_INFO_ACK` primitive in response to a `DL_INFO_REQ` from the user are as follows:

- The maximum SDU is `1500` (`ETHERMTU` - defined in `<sys/ethernet.h>`).
- The minimum SDU is `0`.
- The `DLSAP` address length is `8`.
- The MAC type is `DL_ETHER`.
- The `sap` length value is –2, meaning the physical address component is followed immediately by a 2-byte `sap` component within the `DLSAP` address.
- The broadcast address value is the Ethernet/IEEE broadcast address (`FF:FF:FF:FF:FF:FF`).

Once in the `DL_ATTACHED` state, the user must send a `DL_BIND_REQ` to associate a particular Service Access Point (SAP) with the stream.

Preconfiguration The PCI configuration process varies from system to system. Follow the instructions provided by the vendor.

Known Problems and Limitations

- On multiport cards (exception: Osicom (Rockwell) RNS2340), the first port is the top port. (On the Osicom RNS2340, the first port is the bottom port.)
- If the `dnet` driver fails to determine the correct speed and duplex mode resulting in a corresponding drop in performance, set the speed and duplex mode using the `dnet.conf` file.
- The `dnet` driver incorrectly counts carrier lost or no carrier errors while in full-duplex mode. There is no carrier signal present when in full-duplex mode and it should not be counted as an error.
- Version 4 SROM formats are not supported.

Configuration The `/kernel/drv/dnet.conf` file supports the following options:

`full-duplex` For full duplex operation use `full-duplex=1`, for half duplex use `full-duplex=0`. Half-duplex operation gives better results on older 10mbit networks.

`speed` For 10mbit operation use `speed=10`, for 100mbit operation use `speed=100`. Certain 21140 based cards will operate at either speed. Use the speed property to override the 100mbit default in this case.

Files

`/dev/dnet`	character special device
`/kernel/drv/dnet.conf`	`dnet` configuration file

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86

See Also `attributes(5)`, `dlpi(7P)`, `gld(7D)` `streamio(7I)`

Writing Device Drivers

STREAMS Programming Guide

Name dpt – DPT ServeRAID IV SCSI host bus adapter and RAID adapter driver

Description The `dpt` driver is a driver for the DPT (Distributed Processing Technology) family of SmartRAID IV SCSI HBA and RAID adapters. The following HBA adapters are supported: PM2024, PM2044UW, PM2044W, PM2124, PM2124W, PM2144UW, and PM2144W.

The following RAID adapters are supported: PM3224, PM3224W, PM3334UW, and PM3334W.

Preconfiguration

- DPT PM3224 *only*: Only EPROM 7A and later versions are supported.
- DPT PM2024 and PM2124 *only*: Only EPROM 6D4 and later versions are supported.
- Use adapters with SmartROM version 3.B or later versions only.
- Be sure that the controller board is installed in a PCI bus-mastering slot.
- Disable PCI parity checking if your firmware version is earlier than version 7A, if your system memory is ECC, or if your system does not check parity.

Known Problems and Limitations During system boot, a message may be displayed saying a DPT controller driver cannot be installed. This message indicates that the motherboard installed in your system may contain ECC memory or may not check parity. If you see this message is displayed, disable PCI parity checking.

Supported Settings

- I/O Address: Auto

Configuration Auto-configuration code determines whether the adapter is present at the configured address and what types of devices are attached to it. The DPT ServeRAID is primarily used as a disk array (system drive) controller.

To configure the attached disk arrays, you must configure the controller (using the configuration utilities provided by the hardware manufacturer) before you boot the Solaris operating environment. You use the configuration utilities to set RAID levels, stripe parameters, cache mechanisms and perform other functions. For more information, see the user manual supplied with your hardware.

Files

`/kernel/drv/dpt.conf`	`dpt` configuration file
`/dev/dsk/c`*n*`d`*n*`[s|p]`*n*	block device
`/dev/rdsk/c`*n*`d`*n*`[s|p]`*n*	raw device where:
`c`*n*	controller *n*
`d`*n*	LUN *n* (0–7)
`s`*n*	UNIX system slice *n* (0–15)
`p`*n*	`fdisk(1M)` partition (0)

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86

See Also fdisk(1M), attributes(5), cmdk(7D)

Name dr, drmach, ngdr, ngdrmach – Sun Enterprise 10000 dynamic reconfiguration driver

Synopsis

```
dr

drmach

ngdr

ngdrmach
```

Description The dynamic reconfiguration (DR) driver consists of a platform-independent driver and a platform-specific module. The DR driver uses standard features of the Solaris operating environment whenever possible to control DR operations and calls the platform specific module as needed. The DR driver creates minor nodes in the file system that serve as attachment points for DR operations.

The DR driver provides a pseudo-driver interface to sequence attach and detach operations on system boards using file system entry points referred to as ”attachment points.” The attachment point form depends on the platform.

Sun Enterprise 10000 Server

On the Sun Enterprise 10000 server, the DR driver consists of a platform-independent driver (`ngdr`) and a platform-specific module (`ngdrmach`).

The domain configuration server (DCS) accepts DR requests from the system services processor (SSP) and uses the `libcfgadm`(3LIB) interface to initiate the DR operation. After the operation is performed, the results are returned to the SSP. For more information about the DCS on the Sun Enterprise 10000, refer to the `dcs`(1M) man page and the *Sun Enterprise 10000 Dynamic Reconfiguration User Guide*.

The DR driver creates physical attachment points for system board slots that takes the following form:

```
/devices/pseudo/ngdr@0:SBx
```

Where *x* represents the slot number (0 to 15) for a particular board.

The `cfgadm_sbd`(1M) plugin creates dynamic attachment points that refer to components on system boards, including CPUs, memory, or I/O devices. Refer to the `cfgadm_sbd`(1M) man page for more details.

See Also `cfgadm_sbd`(1M), `ioctl`(2), `libcfgadm`(3LIB)

Sun Enterprise 10000 Dynamic Reconfiguration User Guide

Name dscpmk – Differentiated Services Code Point Marker

Description The `dscpmk` marker is an action module that is executed as a result of classifying or metering packets. It sets a codepoint in the IP header as defined in *RFC-2474: Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*.

Statistics The `dscpmk` module exports the following statistics available through `kstat`:

Global statistics:

```
module: dscpmk                    instance: <action id>
  name: dscpmk stats              class <action name>
        crtime
        snaptime
        npackets                   <number of packets>
        epackets                   <number of packets in error>
        ipackets                   <number of packets not processed>
        dscp_unchanged             <number of packets with DSCP unchanged>
        dscp_changed               <number of packets with DSCP changed>

Also, for each DSCP the following is exported:
     module: dscpmk                         instance: <action id>
     name: dscpmk_dscp0x<DSCP> value class:  <action name>
     dscp                                   <DSCP value>
     npackets                               <number of packets for this DSCP>
```

Files `/kernel/ipp/sparcv9/dscpmk` 64-bit module (SPARC only.)

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWqos

See Also `ipqosconf(1M)`, `dlcosmk(7ipp)`, `flowacct(7ipp)`, `ipqos(7ipp)`, `ipgpc(7ipp)`, `tokenmt(7ipp)`, `tswtclmt(7ipp)`

RFC 2474, Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers K. Nichols, S. Blake, F. Baker, D. Black — The Internet Society, 1998

Name dsp – generic audio device interface

Synopsis

```
#include <sys/soundcard.h>
```

Description To record audio input, applications `open()` the appropriate device and read data from it using the `read()` system call. Similarly, sound data is queued to the audio output port by using the `write(2)` system call. Device configuration is performed using the `ioctl(2)` interface.

Because some systems can contain more than one audio device, application writers are encouraged to open the `/dev/mixer` device and determine the physical devices present on the system using the `SNDCTL_SYSINFO` and `SNDCTL_AUDIOINFO` ioctls. See `mixer(7I)`. The user should be provided a the ability to select a different audio device, or alternatively, an environment variable such as `AUDIODSP` can be used. In the absence of any specific configuration from the user, the generic device file, `/dev/dsp`, can be used. This normally points to a reasonably appropriate default audio device for the system.

Opening the Audio Device

The audio device is not treated as an exclusive resource.

Each `open()` completes as long as there are channels available to be allocated. If no channels are available to be allocated, the call returns `-1` with the `errno` set to `EBUSY`.

Audio applications should explicitly set the encoding characteristics to match the audio data requirements after opening the device, and not depend on any default configuration.

Recording Audio Data

The `read()` system call copies data from the system's buffers to the application. Ordinarily, `read()` blocks until the user buffer is filled. The `poll(2)` system call can be used to determine the presence of data that can be read without blocking. The device can alternatively be set to a non-blocking mode, in which case `read()` completes immediately, but can return fewer bytes than requested. Refer to the `read(2)` manual page for a complete description of this behavior.

When the audio device is opened with read access, the device driver allocates resources for recording. Since this consumes system resources, processes that do not record audio data should open the device write-only (`O_WRONLY`).

The recording process can be stopped by using the `SNDCTL_DSP_HALT_INPUT` ioctl, which also discards all pending record data in underlying device FIFOs.

Before changing record parameters, the input should be stopped using the `SNDCTL_DSP_HALT_INPUT` ioctl, which also flushes the any underlying device input FIFOs. (This is not necessary if the process never started recording by calling `read(2)`. Otherwise, subsequent reads can return samples in the old format followed by samples in the new format. This is particularly important when new parameters result in a changed sample size.

Input data can accumulate in device buffers very quickly. At a minimum, it accumulates at 8000 bytes per second for 8-bit, 8 KHz, mono, `u-Law` data. If the device is configured for more channels, higher sample resolution, or higher sample rates, it accumulates even faster. If the application that consumes the data cannot keep up with this data rate, the underlying FIFOs can become full. When this occurs, any new incoming data is lost until the application makes

room available by consuming data. Additionally, a record overrun is noted, which can be retrieved using the SNDCTL_DSP_GETERROR ioctl.

Record volume for a stream can be adjusted by issuing the SNDCTL_DSP_SETRECVOL ioctl. The volume can also be retrieved using the SNDCTL_DSP_GETRECVOL.

Playing Audio Data

The write() system call copies data from an application's buffer to the device output FIFO. Ordinarily, write() blocks until the entire user buffer is transferred. The device can alternatively be set to a non-blocking mode, in which case write() completes immediately, but might have transferred fewer bytes than requested. See write(2).

Although write() returns when the data is successfully queued, the actual completion of audio output might take considerably longer. The SNDCTL_DSP_SYNC ioctl can be issued to allow an application to block until all of the queued output data has been played.

The final close(2) of the file descriptor waits until all of the audio output has drained. If a signal interrupts the close(), or if the process exits without closing the device, any remaining data queued for audio output is flushed and the device is closed immediately.

The output of playback data can be halted entirely, by calling the SNDCTL_DSP_HALT_OUTPUT ioctl. This also discards any data that is queued for playback in device FIFOs.

Before changing playback parameters, the output should be drained using the SNDCTL_DSP_SYNC ioctl, and then stopped using the SNDCTL_DSP_HALT_OUTPUT ioctl, which also flushes the any underlying device output FIFOs. This is not necessary if the process never started playback, such as by calling write(2). This is particularly important when new parameters result in a changed sample size.

Output data is played from the playback buffers at a default rate of at least 8000 bytes per second for u-Law, A-Law or 8-bit PCM data (faster for 16-bit linear data or higher sampling rates). If the output FIFO becomes empty, the framework plays silence, resulting in audible stall or click in the output, until more data is supplied by the application. The condition is also noted as a play underrun, which can be determined using the SNDCTL_DSP_GETERROR ioctl.

Playback volume for a stream can be adjusted by issuing the SNDCTL_DSP_SETPLAYVOL ioctl. The volume can also be retrieved using the SNDCTL_DSP_GETPLAYVOL.

Asynchronous I/O

The O_NONBLOCK flag can be set using the F_SETFL fcntl(2) to enable non-blocking read() and write() requests. This is normally sufficient for applications to maintain an audio stream in the background.

It is also possible to determine the amount of data that can be transferred for playback or recording without blocking using the SNDCTL_DSP_GETOSPACE or SNDCTL_DSP_GETISPACE ioctls, respectively.

Mixer Pseudo-Device

The `/dev/mixer` provides access to global hardware settings such as master volume settings, etc. It is also the interface used for determining the hardware configuration on the system.

Applications should open(2) `/dev/mixer`, and use the `SNDCTL_SYSINFO` and `SNDCTL_AUDIOINFO` ioctls to determine the device node names of audio devices on the system. See `mixer(7I)` for additional details.

ioctls

Information IOCTLs

The following ioctls are supported on the audio device, as well as the mixer device. See `mixer(7I)` for details.

```
OSS_GETVERSION
SNDCTL_SYSINFO
SNDCTL_AUDIOINFO
SNDCTL_MIXERINFO
SNDCTL_CARDINFO
```

Audio IOCTLs

The `dsp` device supports the following ioctl commands:

`SNDCTL_DSP_SYNC`

The argument is ignored. This command suspends the calling process until the output FIFOs are empty and all queued samples have been played, or until a signal is delivered to the calling process. An implicit `SNDCTL_DSP_SYNC` is performed on the final `close()` of the `dsp` device.

This ioctl should not be used unnecessarily, as if it is used in the middle of playback it causes a small click or pause, as the FIFOs are drained. The correct use of this ioctl is just before changing sample formats.

`SNDCTL_DSP_HALT`
`SNDCTL_DSP_HALT_INPUT`
`SNDCTL_DSP_HALT_OUTPUT`

The argument is ignored. All input or output (or both) associated with the file is halted, and any pending data is discarded.

`SNDCTL_DSP_SPEED`

The argument is a pointer to an integer, indicating the sample rate (in Hz) to be used. The rate applies to both input and output for the file descriptor. On return the actual rate, which can differ from that requested, is stored in the integer pointed to by the argument. To query the configured speed without changing it the value 0 can be used by the application

`SNDCTL_DSP_GETFMTS`

The argument is a pointer to an integer, which receives a bit mask of encodings supported by the device. Possible values are

```
AFMT_MU_LAW      8-bit unsigned u-Law
AFMT_A_LAW       8-bit unsigned a-Law
AFMT_U8          8-bit unsigned linear PCM
AFMT_S16_LE      16-bit signed
                  little-endian linear PCM
AFMT_S16_BE      16-bit signed
                  big-endian linear PCM
AFMT_S16_NE      16-bit signed native-endian
                  linear PCM
AFMT_U16_LE      16-bit unsigned
                  little-endian linear PCM
AFMT_U16_BE      16-bit unsigned big-endian
                  linear PCM
AFMT_U16_NE      16-bit unsigned big-endian
                  linear PCM
AFMT_S24_LE      24-bit signed little-endian
                  linear PCM, 32-bit aligned
AFMT_S24_BE      24-bit signed big-endian
                  linear PCM, 32-bit aligned
AFMT_S24_NE      24-bit signed native-endian
                  linear PCM, 32-bit aligned
AFMT_S32_LE      32-bit signed little-endian
                  linear PCM
AFMT_S32_BE      32-bit signed big-endian
                    linear PCM
AFMT_S32_NE      32-bit signed native-endian
                    linear PCM
AFMT_S24_PACKED  24-bit signed little-endian
                    packed linear PCM
```

Not all devices support all of these encodings. This implementation uses `AFMT_S24_LE` or `AFMT_S24_BE`, whichever is native, internally.

`SNDCTL_DSP_SETFMT` — The argument is a pointer to an integer, which indicates the encoding to be used. The same values as for `SNDCTL_DSP_GETFMT` can be used, but the caller can only specify a single option. The encoding is used for both input and output performed on the file descriptor.

`SNDCTL_DSP_CHANNELS` — The argument is a pointer to an integer, indicating the number of channels to be used (1 for mono, 2 for stereo, etc.) The value applies to both input and output for the file descriptor. On return the actual channel configuration (which can differ from that requested) is stored in the integer

pointed to by the argument. To query the configured channels without changing it the value 0 can be used by the application.

SNDDCTL_DSP_GETCAPS — The argument is a pointer to an integer bit mask, which indicates the capabilities of the device. The bits returned can include

```
PCM_CAP_OUTPUT  Device supports playback
PCM_CAP_INPUT   Device supports recording
PCM_CAP_DUPLEX  Device supports simultaneous
                    playback and recording
```

SNDCTL_DSP_GETPLAYVOL
SNDCTL_DSP_GETRECVOL — The argument is a pointer to an integer to receive the volume level for either playback or record. The value is encoded as a stereo value with the values for two channels in the least significant two bytes. The value for each channel thus has a range of 0-100. In this implementation, only the low order byte is used, as the value is treated as a monophonic value, but a stereo value (with both channel levels being identical) is returned for compatibility.

SNDCTL_DSP_SETPLAYVOL
SNDCTL_DSP_SETRECVOL — The argument is a pointer to an integer indicating volume level for either playback or record. The value is encoded as a stereo value with the values for two channels in the least significant two bytes. The value for each channel has a range of 0-100. Note that in this implementation, only the low order byte is used, as the value is treated as a monophonic value. Portable applications should assign the same value to both bytes

SNDCTL_DSP_GETISPACE
SNDCTL_DSP_GETOSPACE — The argument is a pointer to a `struct audio_buf_info`, which has the following structure:

```
typedef struct audio_buf_info {
   int fragments;* /# of available fragments */
   int fragstotal;
        /* Total # of fragments allocated */
   int fragsize;
        /* Size of a fragment in bytes */
   int bytes;
       /* Available space in bytes */
   /* Note! 'bytes' could be more than
      fragments*fragsize */
} audio_buf_info;
```

The fields fragments, `fragstotal`, and `fragsize` are intended for use with compatible applications (and in the future with `mmap(2)`) only, and need not be used by typical applications. On successful return the bytes member contains the number of bytes that can be transferred without blocking.

`SNDCTL_DSP_CURRENT_IPTR`
`SNDCTL_DSP_CURRENT_OPTR`

The argument is a pointer to an `oss_count_t`, which has the following definition:

```
typedef struct {
    long long samples;
       /* Total # of samples */
    int fifo_samples;
       /* Samples in device FIFO */
    int filler[32];/* For future use */
} oss_count_t;
```

The `samples` field contains the total number of samples transferred by the device so far. The `fifo_samples` is the depth of any hardware FIFO. This structure can be useful for accurate stream positioning and latency calculations.

`SNDCTL_DSP_GETIPTR`
`SNDCTL_DSP_GETOPTR`

The argument is a pointer to a `struct count_info`, which has the following definition:

```
typedef struct count_info {
    unsigned int bytes;
      /* Total # of bytes processed */
    int blocks;
      /* # of fragment transitions since
      last time */
    int ptr;/* Current DMA pointer value */
} count_info;
```

These ioctls are primarily supplied for compatibility, and should not be used by most applications.

`SNDCTL_DSP_GETODELAY`

The argument is a pointer to an integer. On return, the integer contains the number of bytes still to be played before the next byte written are played. This can be used for accurate determination of device latency. The result can differ from actual value by up the depth of the internal device FIFO, which is typically 64 bytes.

`SNDCTL_DSP_GETERROR`

The argument is a pointer to a `struct audio_errinfo`, defined as follows:

```
typedef struct audio_errinfo {
     int play_underruns;
     int rec_overruns;
     unsigned int play_ptradjust;
     unsigned int rec_ptradjust;
     int play_errorcount;
     int rec_errorcount;
     int play_lasterror;
     int rec_lasterror;
     int play_errorparm;
     int rec_errorparm;
     int filler[16];
} audio_errinfo;
```

For this implementation, only the `play_underruns` and `rec_overruns` values are significant. No other fields are used in this implementation.

These fields are reset to zero each time their value is retrieved using this ioctl.

Compatibility IOCTLS

These ioctls are supplied exclusively for compatibility with existing applications. Their use is not recommended, and they are not documented here. Many of these are implemented as simple no-ops.

```
SNDCTL_DSP_POST
SNDCTL_DSP_STEREO
SNDCTL_DSP_SETDUPLEX
SNDCTL_DSP_LOW_WATER
SNDCTL_DSP_PROFILE
SNDCTL_DSP_GETBLKSIZE
SNDCTL_DSP_SUBDIVIDE
SNDCTL_DSP_SETFRAGMENT
SNDCTL_DSP_COOKEDMODE
SNDCTL_DSP_READCTL
SNDCTL_DSP_WRITECTL
SNDCTL_DSP_SILENCE
SNDCTL_DSP_SKIP
SNDCTL_DSP_POST
SNDCTL_DSP_GET_RECSRC
SNDCTL_DSP_SET_RECSRC
SNDCTL_DSP_SET_RECSRC_NAMES
SNDCTL_DSP_GET_PLAYTGT
SNDCTL_DSP_SET_PLAYTGT
SNDCTL_DSP_SET_PLAYTGT_NAMES
SNDCTL_DSP_GETTRIGGER
SNDCTL_DSP_SETTRIGGER
```

```
SNDCTL_AUDIOINFO_EX
SNDCTL_ENGINEINFO
```

Errors An `open()` fails if:

`EBUSY` The requested play or record access isbusy and either the `O_NDELAY` or `O_NONBLOCK` flag was set in the `open()` request.

`EINTR` The requested play or record access is busy and a signal interrupted the `open()` request.

`EINVAL` The device cannot support the requested play or record access.

An `ioctl()` fails if:

`EINVAL` The parameter changes requested in the ioctl are invalid or are not supported by the device.

Files The physical audio device names are system dependent and are rarely used by programmers. Programmers should use the generic device names listed below.

`/dev/dsp`	Symbolic link to the system's primary audio device
`/dev/mixer`	Symbolic link to the pseudo mixer device for the system
`/dev/sndstat`	Symbolic link to the pseudo mixer device for the system
`/usr/share/audio/samples`	Audio files

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86
Availability	SUNWcsu, SUNWaudd, SUNWaudh
Stability Level	Uncommitted

See Also `close(2)`, `fcntl(2)`, `ioctl(2)`, `mmap(2)`, `open(2)`, `poll(2)`, `read(2)`, `write(2)`, `attributes(5)`, `audio(7D)`, `mixer(7I)`

Name dtrace – DTrace dynamic tracing facility

Description The `dtrace` driver provides the dynamic instrumentation and tracing facilities for the DTrace software, as well as the built-in `dtrace` provider. The `dtrace` driver is not a public interface and you access the instrumentation offered by this provider through DTrace tools such as `dtrace(1M)`. Refer to the *Solaris Dynamic Tracing Guide* for a description of the public documented interfaces available for the DTrace facility and dtrace provider probes.

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWdtrp
Interface Stability	Private

See Also `dtrace(1M)`, `libdtrace(3LIB)`, `attributes(5)`

Solaris Dynamic Tracing Guide

Name e1000g, e1000 – Intel PRO/1000 Gigabit family of network interface controllers

Synopsis `/dev/e1000g`

Description The `e1000g` Gigabit Ethernet driver is a multi-threaded, loadable, clonable, GLD-based STREAMS driver supporting the Data Link Provider Interface, `dlpi(7P)`, over Intel PRO/1000 family of Gigabit controllers. This driver supports multiple Intel Gigabit controllers installed within the system. The `e1000g` driver provides basic support including chip initialization, frame transmit and receive, multicast support, and error recovery and reporting.

Application Programming Interface The cloning, character-special device `/dev/e1000g` is used to access all Intel Gigabit devices installed within the system.

The `e1000g` driver is managed by the `dladm(1M)` command line utility, which allows VLANs to be defined on top of e1000g instances and for e1000g instances to be aggregated. See `dladm(1M)` for more details.

You must send an explicit DL_ATTACH_REQ message to associate the opened stream with a particular device (PPA). The PPA ID is interpreted as an unsigned integer data type and indicates the corresponding device instance (unit) number. The driver returns an error (DL_ERROR_ACK) if the PPA field value does not correspond to a valid device instance number.

The values returned by the driver in the `DL_INFO_ACK` primitive in response to the `DL_INFO_REQ` are as follows:

- Maximum SDU (with jumbo frame) is as high as `16298`.
- Minimum SDU is `0`. The driver pads to the mandatory 60-octet minimum packet size.
- The `dlsap` address length is `8`.
- MAC type is `DL_ETHER`.
- The `sap` length value is –2, meaning the physical address component is followed immediately by a 2-byte `sap` component within the `DLSAP` address.
- The broadcast address value is Ethernet/IEEE broadcast address (`FF:FF:FF:FF:FF:FF`).

Configuration The `e1000g` driver does not support the use of shared RAM on the board.

To configure the `e1000g` driver:

- Use `prtconf -v | grep pci8086,[12][01][01][0-F]` to obtain the instance number of the driver.
- Use `ifconfig e1000g`*instance* `plumb` to plumb the controller.
- Use `ifconfig e1000g`*instance* `inet` *ip_address* `netmask + broadcast + -trailers up` to bring up the interface.
- Use the `ping(1M)` command to contact interfaces on the network to verify that the configuration is operational.

Configuration File Options

The following `e1000g.conf` configuration options are supported:

AutoNegAdvertised

This is a bitmap for the speeds advertised during auto-negotiation.

```
Bit     |  7  |  6  |   5   |   4   |  3   |  2   |  1  |  0
Setting| N/A | N/A | 1000F |  N/A  | 100F | 100H | 10F | 10H
```

The adapter only auto-negotiates to a speed that is advertised. For example: `AutoNegAdvertised = 4` causes an adapter to only advertise auto-negotiation at 100 Mbps, full duplex. No other link speeds are accepted or given during auto-negotiation. `AutoNegAdvertised=47` advertises all speeds available, This is the same as using the default setting of `0`.

`0–255` Allowed values

`0` Default

ForceSpeedDuplex

Specify the speed and duplex mode for each instance.

If you set `ForceSpeedDuplex=7,4`, the `e1000g0` is set to auto-negotiate and `e1000g1` is set to 100 Mbps, full duplex. Note that fiber optic ethernet adapters ignore this setting.

Allowed values are:

`1` 10 Megabits per second, Half Duplex.

`2` 10 Megabits per second, Full Duplex.

`3` 100 Megabits per second, Half Duplex.

`4` 100 Megabits per second, Full Duplex.

`7` Auto-negotiate speed and duplex. (Default).

MaxFrameSize

Upper limit on the maximum MTU size the driver allows. All Intel gigabit adapters (except the 82542-based Intel PRO/1000 adapter) allow the configuration of jumbo frames.

For a Intel PRO/1000 adapter that is later than 82571 (including 82571) the maximum MTU accepted by the MAC is 9216. For others, the maximum MTU accepted by the MAC is 16298. Use `ifconfig(1M)` to configure jumbo frames. Using ifconfig with the adapter instance and the mtu argument (`ifconfig e1000g0 mtu 9216`) configures adapter e1000g0 for the maximum allowable jumbo frame size.

Allowed values are:

0 Standard ethernet frames with a MTU equal to `1500`. (Default).

1 Jumbo frames with a maximum MTU of `4010`.

2 Jumbo frames with a maximum MTU of `8106`.

3 Jumbo frames with a maximum MTU of 16298.

FlowControl
Flow control utilizes ethernet XON and unicast and multicast XOFF packets to allow ethernet equipment to slow down the stream of data between two ethernet devices.

Allowed values are:

0 Disable. Packets can get dropped in high-throughput situations, leading to reduced network performance.

1 Receive only.

2 Transmit only.

3 Receive and transmit. (Default).

4 Use adapter's EEPROM-programmed factory default setting.

TbiCompatibilityEnable
You must enable this feature on Intel 82543CG-based copper adapters to operate correctly with TBI mode ethernet hardware.

Allowed values are:

0 Disable.

1 Enable. (Default).

SetMasterSlave
Controls the PHY master/slave setting. Manually forcing master or slave can reduce time needed to link with Planex 08TX and IO data switches. This setting should remain as the hardware default.

Allowed values are:

0 Hardware default. (Default).

1 Force master.

2 Force slave.

3 Force auto.

By default, the following configuration options are not displayed in the e1000g.conf file. Although they are configurable, you should not change these options:

NumRxDescriptors Number of available receive descriptors. Multiple receive descriptors increase receive performance, but decrease available memory.

80–4096 Allowed values.

1024 Default.

NumTxDescriptors — Number of transmit descriptors available to the driver. Mulitiple transmit descriptors increase transmit performance, but decrease available memory.

`80–4096` Allowed values.

`1024` Default.

NumRxFreeList — Number of pre-allocated buffers that the driver can use for received data. Pre-allocating buffers can improve receive performance but decrease available memory.

`60–4096` Allowed values.

`1024` Default.

MaxNumReceivePackets — Maximum number of receive packets that the driver can handle for each interrupt.

CPU utilization can be lowered through more efficient interrupt management. If this value is increased, the time needed by the CPU to process the individual interrupts increases, thereby nullifying any performance gains realized by handling less interrupts.

`0–1024` Allowed values.

`32` Default.

Configuration Options Using dladm(1M)

In addition to the `e1000g.conf` file, you can also use the `dladm(1M)` command to configure the `e1000g` driver.

To view supported configuration parameters, do the following step:

dladm show-linkprop e1000g0

In addition, the current settings of the parameters can be found using **`dladm show-ether`**. Using `dladm(1M)`, you can set the link speed/duplex using the enabled capability parameters supported by the `e1000g` device. Each parameter contains a boolean value that determines if the device enables that mode of operation. The `adv_autoneg_cap` parameter controls auto-negotiation. When `adv_autoneg_cap` is set to 0, the driver forces the mode of operation selected by the first non-zero parameter in priority order as shown below:

```
en_1000fdx_cap          1000Mbps full duplex
en_100fdx_cap           100Mpbs full duplex
en_100hdx_cap           100Mpbs half duplex
en_10fdx_cap            10Mpbs full duplex
en_10hdx_cap            10Mpbs half duplex
```

Note – The link mode of 1000Mbps half duplex is not supported.

Forced link mode of 1000Mbps full duplex is not supported.

Setting all the enabled link capabilities to 0 results in the link being reset to auto-negotiation with full link capabilities advertised.

```
1                       10Mpbs half duplex
2                       10Mpbs full duplex
3                       100Mpbs half duplex
4                       100Mpbs full duplex
```

Files

`dev/e1000g`	Character special device.
`/kernel/drv/e1000g.conf`	Driver configuration file.
`/kernel/drv/sparcv9/e1000g`	64–bit driver binary (SPARC).
`/kernel/drv/e1000g`	32–bit driver binary (x86)
`/kernel/drv/amd64/e1000g`	64–bit driver binary. (x86).

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86

See Also `dladm(1M)`, `ifconfig(1M)`, `kstat(1M)`, `ping(1M)`, `attributes(5)`, `dlpi(7P)`

Intel PRO/1000 Gigabit Adapter Driver Installation Notes for Solaris

Writing Device Drivers

STREAMS Programming Guide

Network Interfaces Guide

Name ecpp – IEEE 1284 compliant parallel port driver

Synopsis

```
#include <sys/types.h>

#include <sys/ecppio.h>

ecpp@unit-address
```

Description The ecpp driver provides a bi-directional interface to *IEEE 1284* compliant devices as well as a forward single-directional interface to Centronics devices. In addition to the Centronics protocol, the ecpp driver supports the *IEEE 1284*Compatibility, Nibble, and ECP protocols. ECPP_COMPAT_MODE and ECPP_CENTRONICS modes of operation have logically identical handshaking protocols, however devices that support ECPP_COMPAT_MODE are *IEEE 1284* compliant devices. *IEEE 1284* compliant devices support at least ECPP_COMPAT_MODE and ECPP_NIBBLE_MODE. Centronics devices support only ECPP_CENTRONICS mode.

By default, ECPP_COMPAT_MODE devices have a strobe handshaking pulse width of 500ns. For this mode, forward data transfers are conducted by DMA. By default, the strobe pulse width for ECPP_CENTRONICS devices is two microseconds. Forward transfers for these devices are managed through PIO. The default characteristics for both ECPP_COMPAT_MODE and ECPP_CENTRONICS devices may be changed through tunable variables defined in ecpp.conf.

The ecpp driver is an *exclusive-use* device, meaning that if the device is already open, subsequent opens fail with EBUSY.

Default Operation Each time the ecpp device is opened, the device is marked as EBUSY and the configuration variables are set to their default values. The write_timeout period is set to 90 seconds.

The driver sets the mode variable according to the following algorithm: The driver initially attempts to negotiate the link into ECPP_ECP_MODE during open(2). If it fails, the driver tries to negotiate into ECPP_NIBBLE_MODE mode. If that fails, the driver operates in ECPP_CENTRONICS mode. Upon successfully opening the device, *IEEE 1284* compliant devices will be left idle in either reverse idle phase of ECPP_ECP_MODE or in ECPP_NIBBLE_MODE. Subsequent calls to write(2) invokes the driver to move the link into either ECPP_COMPAT_MODE or the forward phase of ECPP_ECP_MODE. After the transfer completes, the link returns to idle state.

The application may attempt to negotiate the device into a specific mode or set the write_timeout values through the ECPPIOC_SETPARMS ioctl(2) call. For mode negotiation to be successful, both the host workstation and the peripheral must support the requested mode.

Tunables Characteristics of the ecpp driver may be tuned by the variables described in /kernel/drv/ecpp.conf. These variables are read by the kernel during system startup. To tune the variables, edit the ecpp.conf file and invoke update_drv(1M) to have the kernel read the file again.

Some Centronics peripherals and certain *IEEE 1284* compatible peripherals will not operate with the parallel port operating in a fast handshaking mode. If printing problems occur, set "fast-centronics" and "fast-1284-compatible" to "false." See /kernel/drv/ecpp.conf for more information.

Read/Write Operation

The ecpp driver is a full duplex STREAMS device driver. While an application is writing to an *IEEE 1284* compliant device, another thread may read from it.

Write Operation

A write(2) operation returns the number of bytes successfully written to the stream head. If a failure occurs while a Centronics device is transferring data, the content of the status bits will be captured at the time of the error and can be retrieved by the application program using the BPPIOC_GETERR ioctl(2) call. The captured status information is overwritten each time an attempted transfer or a BPPIOC_TESTIO ioctl(2) occurs.

Read Operation

If a failure or error condition occurs during a read(2), the number of bytes successfully read is returned (short read). When attempting to read a port that has no data currently available, read(2) returns 0 if O_NDELAY is set. If O_NONBLOCK is set, read(2) returns -1 and sets errno to EAGAIN. If O_NDELAY and O_NONBLOCK are clear, read(2) blocks until data become available.

ioctls

The ioctl(2) calls described below are supported. Note that when ecpp is transferring data, the driver waits until the data has been sent to the device before processing the ioctl(2) call.

The ecpp driver supports prnio(7I) interfaces.

Note – The PRNIOC_RESET command toggles the nInit signal for 2 ms, followed by default negotiation.

The following ioctl(2) calls are supported for backward compatibility and are not recommended for new applications:

ECPPIOC_GETPARMS

Get current transfer parameters. The argument is a pointer to a struct ecpp_transfer_parms. See below for a description of the elements of this structure. If no parameters have been configured since the device was opened, the structure will be set to its default configuration. See Default Operation above for more information.

ECPPIOC_SETPARMS

Set transfer parameters. The argument is a pointer to a struct ecpp_transfer_parms. If a parameter is out of range, EINVAL is returned. If the peripheral or host device cannot support the requested mode, EPROTONOSUPPORT is returned. See below for a description of ecpp_transfer_parms and its valid parameters.

The Transfer Parameters Structure is defined in <sys/ecppio.h>.

```
struct ecpp_transfer_parms {
    int  write_timeout;
    int  mode;
};
```

The write_timeout field is set to the value of ecpp-transfer-timeout specified in the ecpp.conf. The write_timeout field specifies how long the driver will wait for the

peripheral to respond to a transfer request. The value must be greater than `0` and less than `ECPP_MAX_TIMEOUT`. All other values are out of range.

The mode field reflects the *IEEE 1284* mode to which the parallel port is currently configured. The mode may be set to one of the following values only: `ECPP_CENTRONICS`, `ECPP_COMPAT_MODE`, `ECPP_NIBBLE_MODE`, `ECPP_ECP_MODE`. All other values are invalid. If the requested mode is not supported, `ECPPIOC_SETPARMS` will return `EPROTONOSUPPORT` and the mode will be set to `ECPP_CENTRONICS` mode. Afterwards, the application may change the mode back to the original mode with `ECPPIOC_SETPARMS`.

`ECPPIOC_GETDEVID` This ioctl gets the *IEEE 1284* device ID from the peripheral in specified mode. Currently, the device ID can be retrieved only in Nibble mode. A pointer to the structure defined in `<sys/ecppsys.h>` must be passed as an argument.

The 1284 device ID structure:

```
struct ecpp_device_id {
  int  mode; /* mode to use for reading device id */
  int  len;  /* length of buffer */
  int  rlen;  /* actual length of device id string */
  char *addr; /* buffer address */
};
```

The mode is the *IEEE 1284* mode into which the port will be negotiated to retrieve device ID information. If the peripheral or host do not support the mode, `EPROTONOSUPPORT` is returned. Applications should set mode to `ECPP_NIBBLE_MODE`. `len` is the length of the buffer pointed to by `addr`. `rlen` is the actual length of the device ID string returned from the peripheral. If the returned `rlen` is greater than `len`, the application can call `ECPPIOC_GETDEVID` again with a buffer length equal or greater than `rlen`. Note that the two length bytes of the *IEEE 1284* device ID are not taken into account and are not returned in the user buffer.

After `ECPPIOC_GETDEVID` successfully completes, the driver returns the link to `ECPP_COMPAT_MODE`. The application is responsible for determining the previous mode the link was operating in and returning the link to that mode.

`BPPIOC_TESTIO` Tests the forward transfer readiness of a peripheral operating in Centronics or Compatibility mode.

TESTIO determines if the peripheral is ready to receive data by checking the open flags and the Centronics status signals. If the current mode of the device is ECPP_NIBBLE_MODE, the driver negotiates the link into ECPP_COMPAT_MODE, check the status signals and then return the link to ECPP_NIBBLE_MODE mode. If the current mode is ECPP_CENTRONICS or ECPP_COMPAT_MODE, TESTIO examines the Centronics status signals in the current mode. To receive data, the device must have the nErr and Select signals asserted and must not have the PE and Busy signals asserted. If ecpp is transferring data, TESTIO waits until the previous data sent to the driver is delivered before executing TESTIO. However if an error condition occurs while a TESTIO is waiting, TESTIO returns immediately. If TESTIO determines that the conditions are ok, 0 is returned. Otherwise, -1 is returned, errno is set to EIO and the state of the status pins is captured. The captured status can be retrieved using the BPPIOC_GETERR ioctl(2) call. The timeout_occurred and bus_error fields will never be set by this ioctl(2). BPPIOC_TESTIO and BPPIOC_GETERR are compatible to the ioctls specified in bpp(7D).

BPPIOC_GETERR

Get last error status. The argument is a pointer to a struct bpp_error_status defined in <sys/bpp_io.h> header file. The error status structure is:

```
struct bpp_error_status {
   char    timeout_occurred; /* 1=timeout */
   char    bus_error;        /* not used */
   uchar_t pin_status;       /* status of pins which
                             /* could cause error */
};
```

The pin_status field indicates possible error conditions. The valid bits for pin_status are: BPP_ERR_ERR, BPP_SLCT_ERR, BPP_PE_ERR, BPP_BUSY_ERR. A set bit indicates that the associated pin is asserted.

This structure indicates the status of all the appropriate status bits at the time of the most recent error condition during a write(2) call, or the status of the bits at the most recent BPPIOC_TESTIO ioctl(2)call.

pin_status indicates possible error conditions under ECPP_CENTRONICS or ECPP_COMPAT_MODE. Under these modes, the state of the status pins will indicate the state of the device. For instance, many Centronics printers lower the nErr signal when a paper jam occurs. The behavior of the status pins depends on the device. Additional status information may be retrieved through the backchannel.

The `timeout_occurred` value is set when a timeout occurs during `write(2)`. `bus_error` is not used in this interface.

The following ioctls are used to directly read and write the parallel port status and control signals. If the current mode of the device is `ECPP_ECP_MODE` or `ECPP_NIBBLE_MODE`, the driver negotiates the link into `ECPP_COMPAT_MODE`, gets or sets the registers and then returns the link to `ECPP_NIBBLE_MODE`. If the current mode is `ECPP_CENTRONICS` or `ECPP_COMPAT_MODE`, these ioctls will get/set the register values in the current mode.

`ECPPIOC_GETREGS` Read register values. The argument is a pointer to a `struct ecpp_regs`. See below for a description of this structure.

`ECPPIOC_SETREGS` Set ecpp register values. The argument is a pointer to a `struct ecpp_regs`. See below for a description of this structure. If a parameter is out of range, `EINVAL` is returned.

The Port Register Structure is defined in <`sys/ecppio.h`>.

```
struct ecpp_regs {
    uchar     dsr;  /* status reg */
    u_char    dcr;  /* control reg */
};
```

The status register is read-only. The `ECPPIOC_SETREGS` ioctl has no affect on this register. Valid bit values for dsr are: `ECPP_nERR`, `ECPP_SLCT`, `ECPP_PE`, `ECPP_nACK`, `ECPP_nBUSY`. All other bits are reserved and always return 1.

The control register is read/write. Valid bit values for dcr are: `ECPP_STB`, `ECPP_AFX`, `ECPP_nINIT`, `ECPP_SLCTIN`. All other bits are reserved. Reading reserved bits always return 1. An attempt to write 0s into these bits results in `EINVAL`.

Device Special Files

`/dev/lp`*N* Solaris x86 only. (Backwards compatibility with former `lp(7D)` devices.)

`/dev/printers/`*N* 1284 compliant parallel port device special files appears in both namespaces.

Files

`kernel/drv/ecpp`	32–bit ELF kernel module
`kernel/drv/sparcv9/ecpp`	64–bit SPARC ELF kernel module
`kernel/drv/amd64/ecpp`	64–bit x86 ELF kernel module
`kernel/drv/ecpp.conf`	driver configuration file
`kernel/drv/sparcv9/ecpp.conf`	driver configuration file for 64–bit SPARC
`kernel/drv/amd64/ecpp.conf`	driver configuration file for 64–bit x86

Errors

`EBADF`	The device is opened for write-only access and a read is attempted, or the device is opened for read-only access and a write is attempted.
`EBUSY`	The device has been opened and another open is attempted. An attempt has been made to unload the driver while one of the units is open.
`EINVAL`	A `ECPPIOC_SETPARMS` `ioctl()` is attempted with an out-of-range value in the `ecpp_transfer_parms` structure. A `ECPPIOC_SETREGS` `ioctl()` is attempted with an invalid value in the `ecpp_regs` structure. An `ioctl()` is attempted with an invalid value in the command argument.An invalid command argument is received during `modload(1M)` or `modunload(1M)`.
`EIO`	The driver encountered a bus error when attempting an access. A read or write did not complete properly, due to a peripheral error or a transfer timeout.
`ENXIO`	The driver has received an open request for a unit for which the attach failed. The driver has received a write request for a unit which has an active peripheral error.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	PCI-based systems
	ISA-based systems (x86)
Availability	SUNWpd (Sparc)
	SUNWpsdcr (x86)
Interface stability	Evolving

See Also `modload(1M)`, `modunload(1M)`, `update_drv(1M)ioctl(2)`, `open(2)`, `read(2)`, `write(2)`, `attributes(5)`, `bpp(7D)`, `usbprn(7D)`, `prnio(7I)`, `streamio(7I)`

IEEE Std 1284–1994

http://www.sun.com/io

Diagnostics

Parallel port controller not supported	Driver does not support parallel port controller on the given host. Attach failed.

Name ehci – Enhanced host controller driver

Synopsis `usb@unit-address`

Description The `ehci` driver is a USBA (Solaris USB Architecture) compliant nexus driver that supports the Enhanced Host Controller Interface Specification 2.0, an industry standard developed by Intel.

A USB 2.0 host controller includes one high-speed host controller and zero or more USB 1.1 host controllers. The high-speed host controller implements an EHCI (Enhanced Host Controller Interface) that is used for all high-speed communications to high-speed-mode devices.

All USB 2.0 devices connected to the root ports of the USB 2.0 host controller and all devices connected to a high- speed-mode hub should be routed to the EHCI host controller.

All full- and low-speed devices connected to the root ports of the USB 2.0 host controller should be routed to the companion USB 1.1 host controllers. (OHCI or UHCI host controller).

The `ehci` supports bulk, interrupt, control and iso chronous transfers (on USB1.*x* devices behind a USB2.0 hub).

Files

`/kernel/drv/ehci`	32–bit ELF 86 kernel module
`/kernel/drv/sparcv9/ehci`	64–bit SPARC ELF kernel module
`/kernel/drv/amd64/ehci`	64–bit x86 ELF kernel module
`/kernel/drv/ehci.conf`	Driver configuration file

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86, PCI-based systems
Availability	SUNWusb

See Also `add_drv(1M)`, `prtconf(1M)`, `rem_drv(1M)`, `update_drv(1M)`, `attributes(5)`, `hubd(7D)`, `uhci(7D)`, `ohci(7D)`, `usba(7D)`

Writing Device Drivers

Universal Serial Bus Specification 2.0

Enhanced Host Controller Interface Specification 1.0

System Administration Guide: Basic Administration

http://www.usb.org

http://www.sun.com/io

http://www.sun.com/bigadmin/hcl

http://www.intel.com/technology/usb/ehcispec.htm

Diagnostics In addition to being logged, the following messages may appear on the system console. All messages are formatted in the following manner:

```
WARNING: <device path> (ehci<instance number>): Message...
```

Unrecoverable USB hardware error.
: There was an unrecoverable USB hardware error reported by the `ehci` controller. Reboot the system. If this problem persists, contact your system vendor.

No SOF interrupts.
: No SOF interrupts have been received. This USB EHCI controller is unusable.

Error recovery failure: Please hotplug the 2.0 hub at <device path>.
: The driver failed to clear 2.0 hub's TT buffer. Remove and reinsert the external USB2.0 hub.

Revision<xx> is not supported.
: High speed USB devices prior to revision 0.95 are not supported.

The following messages may be entered into the system log. They are formatted in the following manner:

```
<device path> (ehci<instance number>): Message...
```

Unable to take control from BIOS. Failure is ignored.
: The driver was unable to take control of the EHCI hardware from the system's BIOS. This failure is ignored. To abort the attach on this take-over failure, comment out a property in ehci.conf. (x86 only).

Unable to take control from BIOS.
: The driver is unable to take control of the EHCI hardware from the system's BIOS and aborts the attach. High speed (USB 2.0) support is disabled. In this case, all USB devices run at full/low speed. Contact your system vendor or your system administror for possible changes in BIOS settings. You can disable a property in `ehci.conf` to ignore this failure. (x86 only.)

Low speed device is not supported.
Full speed device is not supported.
: The driver detected a low or full speed device on its root hub port. Per USB 2.0 specification, the device should be routed to a companion host controller (OHCI or UHCI). However, no attached companion host controller appears to be available. Therefore, low and full speed devices are not supported.

Low speed endpoint's poll interval of <n> ms is below threshold. Rounding up to 8 ms.
Low speed endpoints are limited to polling intervals between 8 ms and 255 ms. If a device reports a polling interval that is less than 8 ms, the driver uses 8 ms instead.

Low speed endpoint's poll interval is greater than 255 ms.
The low speed device's polling interval is out of range. The host controller does not allocate bandwidth for this device. This device is not usable.

Full speed endpoint's poll interval must be between 1 and 255 ms.
The full speed device's polling interval is out of range. The host controller does not allocate bandwidth for this device. This device is not usable.

High speed endpoint's poll interval must be between 1 and 16 units.
The high speed device's polling interval is out of range. The host controller will not allocate bandwidth for this device. This device will not be usable. Refer to the USB specification, revision 2.0 for the unit definition.

ehci_modify_qh_status_bit: Failed to halt qh=<address>.
Error recovery failed. Please disconnect and reinsert all devices or reboot.

Note – Due to recently discovered incompatibilities with this USB controller, USB2.*x* transfer support has been disabled. However, this device continues to function as a USB1.*x* controller. Information on enabling USB2.x support is provided in this man page. Please refer to *www.sun.com/io* for Solaris Ready products and to *www.sun.com/bigadmin/hcl* for additional compatible USB products.

VIA chips may not be compatible with this driver. To bind `ehci` specifically to the chip and eliminate the warnings, and to enable USB2.x suppport, a new, more specific driver alias (refer to `add_drv(1M)` and `update_drv(1M)`) must be specified for `ehci`. By default, the `ehci` alias is 'pciclass,0c0320.' The compatible names in the `prtconf(1M)` output provides additional aliases. For example:

```
   # prtconf -vp | grep pciclass,0c0320
        compatible: 'pci1106,3104.1106.3104.2063' +
'pci1106,3104.1106.3104' + 'pci1106,3104' +
 pci1106,3104.2063' + 'pci1106,3104' + 'pciclass,0c0320' +
'pciclass,0c03'
         ....


       A more specific alias is 'pci1106,3104.' Perform the follow-
       ing step to add this alias, then reboot the system:


       # update_drv -a -i '"pci1106,3104"' ehci

       # reboot
```

After you apply the above workaround, the following message is displayed in your system log:

Applying VIA workarounds.

Name elxl – 3Com Ethernet device driver

Synopsis `/kernel/drv/elxl`

Description The `elxl` driver currently supports the following network cards: EtherLink XL (3C900-TPO, 3C900-COMBO, 3C900B-TPO, 3C900B-COMBO, and 3C900B-TPC), EtherLink XL 10/100 (3C905-TX Fast, 3C905-T4 Fast, 3C905B-TX Fast, 3C905B-T4 Fast, and 3C905C-TX-M Fast), and EtherLink Server 10/100 (3C980-TX Fast and 3C980C-TXM).

The `elxl` Ethernet driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface, `dlpi(7P)`. Multiple EtherLink XL controllers installed within the system are supported by the driver. The `elxl` driver provides basic support for the EtherLink hardware. Functions include chip initialization, frame transmit and receive, multicast and promiscuous mode support, and error recovery and reporting.

The cloning, character-special device `/dev/elxl` is used to access all EtherLink devices installed within the system.

The `elxl` driver is dependent on `/kernel/misc/gld`, a loadable kernel module that provides the `elxl` driver with the DLPI and STREAMS functionality required of a LAN driver. See `gld(7D)` for more details on the primatives supported by the driver.

The values returned by the driver in the `DL_INFO_ACK` primitive in response to the `DL_INFO_REQ` from the user are as follows:

- Maximum `SDU` is 1500 (`ETHERMTU`).
- Minimum `SDU` is 0. The driver will pad to the mandatory 60-octet minimum packet size.
- The `dlsap` address length is 8.
- `MAC` type is `DL_ETHER`.
- The `sap` length value is –2, meaning the physical address component is followed immediately by a 2-byte `sap` component within the DLSAP address.
- The broadcast address value is Ethernet/IEEE broadcast address (`FF:FF:FF:FF:FF:FF`).

Preconfiguration

Supported Settings

- Media Type: Auto Select

Known Problems and Limitations

- 3C905B cards in a Compaq ProLiant 6500 can fail to generate interrupts. There is no known workaround for this problem. However, because some slots appear to be more prone to the problem than others, try correcting the problem by moving the card to another PCI slot. If that fails, try rebooting the machine a number of times to free the card from the wedged state.
- Early versions of the 3Com 3C905C-TX-M adapter firmware do not support PXE network boot on Solaris systems. If you are using a version earlier than 4.11, upgrade the firmware. The PXE version is indicated by the Managed Boot Agent version number. This number is not normally displayed during boot, but is shown on the PXE configuration screen.

Configuration The `/kernel/drv/elxl.conf` file supports the following tunable properties that you can set in the `elxl.conf` file:

`tx_start_thresh`	Minimum number of bytes in transmit FIFO (if lower then the packet size), before NIC starts the frame transfer. The higher the value, the less chance for a transmission underrun error that triggers a transmission retry. Default value is 250, maximum value is 1514.
`recv-descriptors`	The number of frames the device may receive from the network without system attention before the device begins dropping the frames. Valid range is 1 to 256. Default is 24.
`xmit-descriptors`	The number of outgoing frames that the system can queue on the driver queue before blocking a send request. Valid range is 1 to 512. Default is 128.
`min-recv-data-buffers`	Minimal number of allocated receive data buffers. Must be equal or greater than effective number of recv-descriptors. Default (max value) is 128.
`max-recv-data-buffers`	Maximum number of allocated receive data buffers. Must be equal or greater then min-recv-data-buffers number.
`inter-frame-space`	Amount of time in addition to the standard inter-frame time used by *IEEE 802.3* deference rule. Expressed in 32 multiples of bit time. Used only in half-duplex context. This small IFS is to make the defer-after-transmit time slightly longer than the defer-after-receive time. This greatly reduces the high collision rate seen with heavy TCP traffic (almost exactly one collision per ACK), giving slightly improved overall throughput in half-duplex mode. Default IFS is 32. This property is ignored in full-duplex mode.
`full-duplex`	When set, it forces full-duplex mode for NIC capable of working in full-duplex mode. If this property is set, the auto-negotiation feature (if supported) of the network controller is disabled. For full duplex operation use `full-duplex=1`. For half duplex use `full-duplex=0`. Half-duplex operation provides better results on older 10-Mbit networks.
`speed`	Link speed in Mbps. Valid values are 10 and 100. If this property is set, the auto-negotiation feature of the network controller (if supported) is disabled.

Files

`/dev/elxl`	Special character device
`/kernel/drv/elxl.conf`	Configuration file for `elxl` driver

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86

See Also `attributes(5)`, `gld(7D)`, `streamio(7I)`, `dlpi(7P)`.

Name emlxs – Emulex-Sun LightPulse Fibre Channel host bus adapter driver

Synopsis `SUNW,emlxs`

Description The `emlxs` host bus adapter driver is a Sun Fibre Channel transport layer-compliant nexus driver for the Emulex Light-Pulse family of Fibre Channel adapters. These adapters support Fibre Channel SCSI and IP Protocols, FC-AL public loop profile, point-to-point fabric connection and Fibre Channel service classes two and three.

The `emlxs` driver interfaces with the Sun Fibre Channel transport layer to support the standard functions provided by the SCSA interface. It supports auto request sense and tagged queueing by default. The driver requires that all devices have unique hard addresses in private loop configurations. Devices with conflicting hard addresses are not accessible.

Files

`/kernel/drv/emlxs`	32–bit ELF kernel module.
`/kernel/drv/amd/emlxs`	64-bit ELF kernel module (x86).
`/kernel/drv/sparcv9/emlxs`	64-bit ELF kernel module (SPARC).
`/kernel/drv/emlxs.conf`	Driver configuration file

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86
Availability	SUNWemlxs

See Also `prtconf(1M)`, `driver.conf(4)`, `fcp(7D)`, `fp(7d)`

Writing Device Drivers

ANSI X3.230:1994, Fibre Channel Physical Signaling (FC-PH)

Project 1134-D, Fibre Channel Generic Services (FC-GS-2)

ANSI X3.269-1996, Fibre Channel Arbitrated Loop (FC-AL)

ANSI X3.270-1996, Fibre Channel Protocol for SCSI (FCP-SCSI)

ANSI X3.270-1996, SCSI-3 Architecture Model (SAM)

Fibre Channel Private Loop SCSI Direct Attach (FC-PLDA)

Fabric Loop Attachment (FC-FLA)

Name eri – eri Fast-Ethernet device driver

Synopsis `/dev/eri`

Description The `eri` Fast Ethernet driver is a multi-threaded, loadable, clonable, STREAMS—based hardware driver supporting the connectionless Data Link Provider Interface `dlpi(7P)` over an `eri` Fast-Ethernet controller. Multiple `eri` devices installed within the system are supported by the driver.

The `eri` driver provides basic support for the `eri` hardware and handles the `eri` device. Functions include chip initialization, frame transit and receive, multicast and promiscuous support, and error recovery and reporting.

The `eri` device provides 100Base-TX networking interfaces using the SUN `RIO` ASIC and an internal transceiver. The `RIO` ASIC provides the PCI interface and MAC functions. The physical layer functions are provided by the internal transceiver which connects to a RJ-45 connector.

The 100Base-TX standard specifies an auto-negotiation protocol to automatically select the mode and speed of operation. The internal transceiver is capable of performing auto-negotiation using the remote-end of the link (link partner) and receives the capabilities of the remote end. It selects the highest common denominator mode of operation based on the priorities. It also supports a forced-mode of operation under which the driver selects the mode of operation.

Application Programming Interface The cloning character-special device `/dev/eri` is used to access all `eri` controllers installed within the system.

eri and DLPI The `eri` driver is a "style 2" Data Link Service provider. All `M_PROTO` and `M_PCPROTO` type messages are interpreted as `DLPI` primitives. Valid `DLPI` primitives are defined in `<sys/dlpi.h>`. Refer to `dlpi(7P)` for more information.

An explicit `DL_ATTACH_REQ` message by the user is required to associate the opened stream with a particular device (`ppa`). The `ppa` ID is interpreted as an `unsigned integer` data type and indicates the corresponding device instance (unit) number. An error (`DL_ERROR_ACK`) is returned by the driver if the `ppa` field value does not correspond to a valid device instance number for this system. The device is initialized on first attach and de-initialized (stopped) at last detach.

The values returned by the driver in the `DL_INFO_ACK` primitive in response to the `DL_INFO_REQ` from the user are as follows:

- The maximum SDU is `1500` (`ETHERMTU` - defined in `<sys/ethernet.h>`).
- The minimum SDU is `0`.
- The `dlsap` address length is `8`.
- The MAC type is `DL_ETHER`.

- The sap length values is –2, meaning the physical address component is followed immediately by a 2 byte sap component within the DLSAP address.
- The service mode is DL_CLDLS.
- Optional quality of service (QOS) is not currently supported so QOS fields are 0.
- The provider style is DL_STYLE.
- The version is DL_VERSION_2.
- The broadcast address value is Ethernet/IEEE broadcast address (0xFFFFFF).

Once in the DL_ATTACHED state, the user must send a DL_BIND_REQ to associate a particular SAP (Service Access Pointer) with the stream. The eri driver interprets the sap field within the DL_BIND_REQ as an Ethernet "type," therefore valid values for the sap field are in the [0-0xFFFF] range. Only one Ethernet type can be bound to the stream at any time.

If the user selects a sap with a value of 0, the receiver will be in IEEE 802.3 mode. All frames received from the media having a Ethernet type field in the range [0-1500] are assumed to be 802.3 frames and are routed up all open Streams which are bound to sap value 0. If more than one Stream is in 802.3 mode, the frame will be duplicated and routed up multiple Streams as DL_UNITDATA_IND messages.

In transmission, the driver checks the sap field of the DL_BIND_REQ to determine if the value is 0 or if the Ethernet type field is in the range [0-1500]. If either is true, the driver computes the length of the message, not including initial M_PROTO mblk (message block), of all subsequent DL_UNITDATA_REQ messages, and transmits 802.3 frames that have this value in the MAC frame header length field.

The eri driver's DLSAP address format consists of the 6 byte physical (Ethernet) address component followed immediately by the 2 byte sap (type) component, producing an 8 byte DLSAP address. Applications should *not* hardcode to this particular implementation-specific DLSAP address format but use information returned in the DL_INFO_ACK primitive to compose and decompose DLSAP addresses. The sap length, full DLSAP length, and sap/physical ordering are included within the DL_INFO_ACK. The physical address length can be computed by subtracting the sap length from the full DLSAP address length or by issuing the DL_PHYS_ADDR_REQ to obtain the current physical address associated with the stream.

Once in the DL_BOUND state, the user may transmit frames on the Ethernet by sending DL_UNITDATA_REQ messages to the eri driver. The eri driver will route received Ethernet frames up all open and bound streams having a sap which matches the Ethernet type as DL_UNITDATA_IND messages. Received Ethernet frames are duplicated and routed up multiple open streams if necessary. The DLSAP address contained within the DL_UNITDATA_REQ and DL_UNITDATA_IND messages consists of both the sap (type) and physical (Ethernet) components.

eri Primitives

In addition to the mandatory connectionless DLPI message set, the driver also supports the following primitives:

The `DL_ENABMULTI_REQ` and `DL_DISABMULTI_REQ` primitives enable/disable reception of individual multicast group addresses. A set of multicast addresses may be iteratively created and modified on a per-stream basis using these primitives. These primitives are accepted by the driver in any state following `DL_ATTACHED`.

The `DL_PROMISCON_REQ` and `DL_PROMISCOFF_REQ` primitives with the `DL_PROMISC_PHYS` flag set in the `dl_level` field enables/disables reception of all promiscuous mode frames on the media, including frames generated by the local host. When used with the `DL_PROMISC_SAP` flag set, this enables/disables reception of all `sap` (Ethernet type) values. When used with the `DL_PROMISC_MULTI` flag set, this enables/disables reception of all multicast group addresses. The effect of each is always on a per-stream basis and independent of the other `sap` and physical level configurations on this stream or other streams.

The `DL_PHYS_ADDR_REQ` primitive returns the 6 octet Ethernet address currently associated (attached) to the stream in the `DL_PHYS_ADDR_ACK` primitive. This primitive is valid only in states following a successful `DL_ATTACH_REQ`.

The `DL_SET_PHYS_ADDR_REQ` primitive changes the 6 octet Ethernet address currently associated (attached) to this stream. The credentials of the process which originally opened this stream must be superuser, or `EPERM` is returned in the `DL_ERROR_ACK`. This primitive is destructive because it affects all current and future streams attached to this device. An `M_ERROR` is sent up all other streams attached to this device when this primitive is successful on this stream. Once changed, all streams subsequently opened and attached to this device will obtain this new physical address. Once changed, the physical address will remain until this primitive is used to change the physical address again or the system is rebooted, whichever comes first.

eri DRIVER

By default, the eri driver performs auto-negotiation to select the mode and speed of the link, which can be in one of the following modes, as described in the 100Base-TX standard:

- 100 Mbps, full-duplex
- 100 Mbps, half-duplex
- 10 Mbps, full-duplex
- 10 Mbps, half-duplex

The auto-negotiation protocol automatically selects:

- Operation mode (half-duplex or full-duplex)
- Speed (100 Mbps or 10 Mbps)

The auto–negotiation protocol does the following:

- Gets all modes of operation supported by the link partner
- Advertises its capabilities to the Link Partner
- Selects the highest common denominator mode of operation based on the priorities

The internal transceiver is capable of all of the operating speeds and modes listed above. By default, auto-negotiation is used to select the speed and the mode of the link and the common mode of operation with the link partner.

For users who want to select the speed and mode of the link, the `eri` device supports programmable IPG (Inter-Packet Gap) parameters `ipg1` and `ipg2`. Sometimes, the user may want to alter these values depending on whether the driver supports 10 Mbps or 100 Mpbs and accordingly, `IPG` will be set to 9.6 or 0.96 microseconds.

eri Parameter List

The `eri` driver provides for setting and getting various parameters for the `eri` device. The parameter list includes current transceiver status, current link status, inter-packet gap, local transceiver capabilities and link partner capabilities.

The local transceiver has two set of capabilities: one set reflects hardware capabilities, which are read-only `(RO)` parameters. The second set reflects the values chosen by the user and is used in speed selection and possess read/write `(RW)` capability. At boot time, these two sets of capabilities will be the same. Because the current default value of these parameters can only be read and not modified, the link partner capabilities are also read only.

Files

`/dev/eri`	`eri` special character device.
`/kernel/drv/eri.conf`	System wide default device driver properties
`/kernel/drv/sparcv9/eri`	64 bit device driver

See Also

`ndd`(1M), `netstat`(1M), `driver.conf`(4), `hme`(7D), `qfe`(7d), `dlpi`(7P)

Name esp – ESP SCSI Host Bus Adapter Driver

Synopsis `esp@`*sbus-slot*`,80000`

Description The `esp` Host Bus Adapter driver is a SCSA compliant nexus driver that supports the Emulex family of esp SCSI chips (esp100, esp100A, esp236, fas101, fas236).

The `esp` driver supports the standard functions provided by the SCSA interface. The driver supports tagged and untagged queuing, fast SCSI (on FAS esp's only), almost unlimited transfer size (using a moving DVMA window approach), and auto request sense; but it does not support linked commands.

Configuration The `esp` driver can be configured by defining properties in `esp.conf` which override the global SCSI settings. Supported properties are: `scsi-options`, `target`<*n*>`-scsi-options`, `scsi-reset-delay`, `scsi-watchdog-tick`, `scsi-tag-age-limit`, `scsi-initiator-id`.

`target`<*n*>`-scsi-options` overrides the `scsi-options` property value for `target`<*n*>. <*n*> can vary from `0` to `7`.

Refer to `scsi_hba_attach(9F)` for details.

Examples **EXAMPLE 1** A sample of esp configuration file.

Create a file `/kernel/drv/esp.conf` and add this line:

```
scsi-options=0x78;
```

This will disable tagged queuing, fast SCSI, and Wide mode for all `esp` instances. To disable an option for one specific `esp` (refer to `driver.conf(4)`):

```
name="esp"
parent="/iommu@f,e0000000/sbus@f,e0001000/espdma@f,400000"
       reg=0xf,0x800000,0x40
       target1-scsi-options=0x58
       scsi-options=0x178 scsi-initiator-id=6;
```

Note that the default initiator ID in OBP is 7 and that the change to ID 6 will occur at attach time. It may be preferable to change the initiator ID in OBP.

The above would set `scsi-options` for target 1 to `0x58` and for all other targets on this SCSI bus to `0x178`. The physical pathname of the parent can be determined using the `/devices` tree or following the link of the logical device name:

```
example# ls -l /dev/rdsk/c0t3d0s0
lrwxrwxrwx   1 root   root   88 Aug 22 13:29 /dev/rdsk/c0t3d0s0 ->
../../devices/iommu@f,e0000000/sbus@f,e0001000/espdma@f,400000/
              esp@f,800000/sd@3,0:a,raw
```

The register property values can be determined from `prtconf(1M)` output (`-v` option):

EXAMPLE 1 A sample of esp configuration file. *(Continued)*

```
esp, instance #0
....
            Register Specifications:
             Bus Type=0xf, Address=0x800000, Size=40
```

To set scsi-options more specifically per target:

```
target1-scsi-options=0x78;
device-type-scsi-options-list =
        "SEAGATE ST32550W", "seagate-scsi-options" ;
seagate-scsi-options = 0x58;
scsi-options=0x3f8;
```

The above would set `scsi-options` for target 1 to `0x78` and for all other targets on this SCSI bus to `0x378` except for one specific disk type which will have `scsi-options` set to 0x58.

`scsi-options` specified per target ID has the highest precedence, followed by `scsi-options` per device type. To get the inquiry string run probe-scsi or probe-scsi-all command at the ok prompt before booting the system.

Global, for example. for all esp instances, `scsi-options` per bus has the lowest precedence.

The system needs to be rebooted before the specified `scsi-options` take effect.

Files

`/kernel/drv/esp`	ELF Kernel Module
`/kernel/drv/esp.conf`	Configuration file

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SBus-based systems with esp-based
	SCSI port and SSHA, SBE/S, FSBE/S,
	and DSBE/S SBus SCSI Host Adapter options

See Also `prtconf(1M)`, `driver.conf(4)`, `attributes(5)`, `fas(7D)`, `scsi_abort(9F)`, `scsi_hba_attach(9F)`, `scsi_ifgetcap(9F)`, `scsi_reset(9F)`, `scsi_sync_pkt(9F)`, `scsi_transport(9F)`, `scsi_device(9S)`, `scsi_extended_sense(9S)`, `scsi_inquiry(9S)`, `scsi_pkt(9S)`

Writing Device Drivers

OpenBoot Command Reference

ANSI Small Computer System Interface-2 (SCSI-2)

ESP Technical Manuals, QLogic Corp.

Diagnostics The messages described below are some that may appear on the system console, as well as being logged.

The first four messages may be displayed while the `esp` driver is trying to attach; these messages mean that the `esp` driver was unable to attach. All of these messages are preceded by "esp%d", where "%d" is the instance number of the `esp` controller.

`Device in slave-only slot`
The SBus device has been placed in a slave-only slot and will not be accessible; move to non-slave-only SBus slot.

`Device is using a hilevel intr`
The device was configured with an interrupt level that cannot be used with this `esp` driver. Check the SBus device.

`Unable to map registers`
Driver was unable to map device registers; check for bad hardware. Driver did not attach to device; SCSI devices will be inaccessible.

`Cannot find dma controller`
Driver was unable to locate a dma controller. This is an auto-configuration error.

`Disabled TQ since disconnects are disabled`
Tagged queuing was disabled because disconnects were disabled in `scsi-options`.

`Bad clock frequency- setting 20mhz, asynchronous mode`
Check for bad hardware.

`Sync pkt failed`
Syncing a SCSI packet failed. Refer to `scsi_sync_pkt(9F)`.

`Slot %x: All tags in use!!!`
The driver could not allocate another tag number. The target devices do not properly support tagged queuing.

`Target %d.%d cannot alloc tag queue\n`
The driver could not allocate space for tag queue.

`Gross error in esp status (%x)`
The driver experienced severe SCSI bus problems. Check cables and terminator.

`Spurious interrupt`
The driver received an interrupt while the hardware was not interrupting.

`Lost state in phasemanage`
The driver is confused about the state of the SCSI bus.

`Unrecoverable DMA error during selection`
The DMA controller experienced host SBus problems. Check for bad hardware.

`Bad sequence step (0x%x) in selection`
The esp hardware reported a bad sequence step. Check for bad hardware.

`Undetermined selection failure`
The selection of a target failed unexpectedly. Check for bad hardware.

`>2 reselection IDs on the bus`
Two targets selected simultaneously, which is illegal. Check for bad hardware.

`Reconnect: unexpected bus free`
A reconnect by a target failed. Check for bad hardware.

`Timeout on receiving tag msg`
Suspect target f/w failure in tagged queue handling.

`Parity error in tag msg`
A parity error was detected in a tag message. Suspect SCSI bus problems.

`Botched tag`
The target supplied bad tag messages. Suspect target f/w failure in tagged queue handling.

`Parity error in reconnect msg's`
The reconnect failed because of parity errors.

`Target <n> didn't disconnect after sending <message>`
The target unexpectedly did not disconnect after sending <message>.

`No support for multiple segs`
The `esp` driver can only transfer contiguous data.

`No dma window?`
Moving the DVMA window failed unexpectedly.

`No dma window on <type> operation`
Moving the DVMA window failed unexpectedly.

`Cannot set new dma window`
Moving the DVMA window failed unexpectedly.

`Unable to set new window at <address> for <type> operation`
Moving the DVMA window failed unexpectedly.

`Illegal dma boundary? %x`
An attempt was made to cross a boundary that the driver could not handle.

`Unwanted data out/in for Target <n>`
The target went into an unexpected phase.

`Spurious <name> phase from target <n>`
The target went into an unexpected phase.

`SCSI bus DATA IN phase parity error`
The driver detected parity errors on the SCSI bus.

`SCSI bus MESSAGE IN phase parity error`
The driver detected parity errors on the SCSI bus.

`SCSI bus STATUS phase parity error`
The driver detected parity errors on the SCSI bus.

`Premature end of extended message`
An extended SCSI bus message did not complete. Suspect a target f/w problem.

`Premature end of input message`
A multibyte input message was truncated. Suspect a target f/w problem.

`Input message botch`
The driver is confused about messages coming from the target.

`Extended message <n> is too long`
The extended message sent by the target is longer than expected.

`<name> message <n> from Target <m> garbled`
Target <m> sent message <name> of value <n> which the driver did not understand.

`Target <n> rejects our message <name>`
Target <n> rejected a message sent by the driver.

`Rejecting message <name> from Target <n>`
The driver rejected a message received from target <n>

`Cmd dma error`
The driver was unable to send out command bytes.

`Target <n> refused message resend`
The target did not accept a message resend.

`Two-byte message <name> <value> rejected`
The driver does not accept this two-byte message.

`Unexpected selection attempt`
An attempt was made to select this host adapter by another initiator.

`Polled cmd failed (target busy)`
A polled command failed because the target did not complete outstanding commands within a reasonable time.

`Polled cmd failed`
A polled command failed because of timeouts or bus errors.

`Disconnected` command `timeout` for `Target <id>.<lun>`
A timeout occurred while target/lun was disconnected. This is usually a target f/w problem. For tagged queuing targets, <n> commands were outstanding when the timeout was detected.

`Disconnected tagged cmds (<n>) timeout for Target <id>.<lun>`
A timeout occurred while target/lun was disconnected. This is usually a target f/w problem. For tagged queuing targets, <n> commands were outstanding when the timeout was detected.

`Connected command timeout for Target <id>.<lun>`
This is usually a SCSI bus problem. Check cables and termination.

`Target <id>.<lun> reverting to async. mode`
A data transfer hang was detected. The driver attempts to eliminate this problem by reducing the data transfer rate.

`Target <id>.<lun> reducing sync. transfer rate`
A data transfer hang was detected. The driver attempts to eliminate this problem by reducing the data transfer rate.

`Reverting to slow SCSI cable mode`
A data transfer hang was detected. The driver attempts to eliminate this problem by reducing the data transfer rate.

`Reset SCSI bus failed`
An attempt to reset the SCSI bus failed.

`External SCSI bus reset`
Another initiator reset the SCSI bus.

Warnings The `esp` hardware does not support Wide SCSI mode. Only FAS-type esp's support fast SCSI (10 MB/sec).

Notes The `esp` driver exports properties indicating per target the negotiated transfer speed (`target<n>-sync-speed`) and whether tagged queuing has been enabled (`target<n>-TQ`). The `sync-speed` property value is the data transfer rate in KB/sec. The `target-TQ` property has no value. The existence of the property indicates that tagged queuing has been enabled. Refer to `prtconf(1M)` (verbose option) for viewing the `esp` properties.

```
        dma, instance #3
            Register Specifications:
                Bus Type=0x2, Address=0x81000, Size=10
            esp, instance #3
                 Driver software properties:
                     name <target3-TQ> length <0> — <no
value>.
                     name <target3-sync-speed> length <4>
                         value <0x00002710>.
                     name <scsi-options> length <4>
                         value <0x000003f8>.
                     name <scsi-watchdog-tick> length <4>
                         value <0x0000000a>.
                     name <scsi-tag-age-limit> length <4>
                         value <0x00000008>.
```

```
name <scsi-reset-delay> length <4>
    value <0x00000bb8>.
```

Name fas – FAS SCSI Host Bus Adapter Driver

Synopsis `fas@`*sbus-slot*`,0x8800000`

Description The `fas` Host Bus Adapter driver is a SCSA compliant nexus driver that supports the Qlogic FAS366 SCSI chip.

The `fas` driver supports the standard functions provided by the SCSA interface. The driver supports tagged and untagged queuing, wide and fast SCSI, almost unlimited transfer size (using a moving DVMA window approach), and auto request sense; but it does not support linked commands.

Driver Configuration The `fas` driver can be configured by defining properties in `fas.conf` which override the global SCSI settings. Supported properties are: `scsi-options`, `target<`*n*`>-scsi-options`, `scsi-reset-delay`, `scsi-watchdog-tick`, `scsi-tag-age-limit`, `scsi-initiator-id`.

`target<`*n*`>-scsi-options` overrides the `scsi-options` property value for `target<`*n*`>`. <*n*> can vary from decimal `0` to `15`. The supported `scsi-options` are: `SCSI_OPTIONS_DR`, `SCSI_OPTIONS_SYNC`, `SCSI_OPTIONS_TAG`, `SCSI_OPTIONS_FAST`, and `SCSI_OPTIONS_WIDE`.

After periodic interval `scsi-watchdog-tick`, the `fas` driver searches all current and disconnected commands for timeouts.

`scsi-tag-age-limit` is the number of times that the `fas` driver attempts to allocate a particular tag ID that is currently in use after going through all tag IDs in a circular fashion. After finding the same tag ID in use `scsi-tag-age-limit` times, no more commands will be submitted to this target until all outstanding commands complete or timeout.

Refer to `scsi_hba_attach(9F)` for details.

Examples **EXAMPLE 1** A sample of fas configuration file

Create a file called `/kernel/drv/fas.conf` and add this line:

```
scsi-options=0x78;
```

This disables tagged queuing, Fast SCSI, and Wide mode for all `fas` instances. The following example disables an option for one specific `fas` (refer to `driver.conf(4)` for more details):

```
name="fas" parent="/iommu@f,e0000000/sbus@f,e0001000"
       reg=3,0x8800000,0x10,3,0x8810000,0x40
       target1-scsi-options=0x58
       scsi-options=0x178 scsi-initiator-id=6;
```

Note that the default initiator ID in OBP is 7 and that the change to ID 6 will occur at attach time. It may be preferable to change the initiator ID in OBP.

The example above sets `scsi-options` for target 1 to `0x58` and all other targets on this SCSI bus to `0x178`.

EXAMPLE 1 A sample of fas configuration file *(Continued)*

The physical pathname of the parent can be determined using the `/devices` tree or following the link of the logical device name:

```
 # ls -l /dev/rdsk/c1t3d0s0
 lrwxrwxrwx 1 root  other  78 Aug 28 16:05 /dev/rdsk/c1t3d0s0 ->

. . /. . /devices/iommu@f,e0000000\
     sbus@f,e0001000/SUNW,fas@3,8800000/sd@3,0:a,raw
```

Determine the register property values using the output from `prtconf(1M)` (with the `-v` option):

```
SUNW,fas, instance #0
   . . . .
  Register Specifications:
      Bus Type=0x3, Address=0x8800000, Size=10
      Bus Type=0x3, Address=0x8810000, Size=40
```

`scsi-options` can also be specified per device type using the device inquiry string. All the devices with the same inquiry string will have the same `scsi-options` set. This can be used to disable some `scsi-options` on all the devices of the same type.

```
device-type-scsi-options-list=
    "TOSHIBA    XM5701TASUN12XCD", "cd-scsi-options";
cd-scsi-options = 0x0;
```

The above entry in `/kernel/drv/fas.conf` sets the `scsi-options` for all devices with inquiry string `TOSHIBA XM5701TASUN12XCD` to `cd-scsi-options`. To get the inquiry string, run the `probe-scsi` or `probe-scsi-all` command at the `ok` prompt before booting the system.

To set `scsi-options` more specifically per target:

```
target1-scsi-options=0x78;
device-type-scsi-options-list =
    "SEAGATE ST32550W", "seagate-scsi-options" ;
seagate-scsi-options = 0x58;
scsi-options=0x3f8;
```

The above sets `scsi-options` for target 1 to `0x78` and for all other targets on this SCSI bus to `0x3f8` except for one specific disk type which will have `scsi-options` set to `0x58`.

`scsi-options` specified per target ID have the highest precedence, followed by `scsi-options` per device type. Global `fas scsi-options` (effecting all instances) per bus have the lowest precedence.

The system needs to be rebooted before the specified `scsi-options` take effect.

Driver Capabilities

The target driver needs to set capabilities in the `fas` driver in order to enable some driver features. The target driver can query and modify these capabilities: `synchronous`, `tagged-qing`, `wide-xfer`, `auto-rqsense`, `qfull-retries`, `qfull-retry-interval`. All other capabilities can only be queried.

By default, `tagged-qing`, `auto-rqsense`, and `wide-xfer` capabilities are disabled, while `disconnect`, `synchronous`, and `untagged-qing` are enabled. These capabilities can only have binary values (`0` or `1`). The default value for `qfull-retries` is `10` and the default value for `qfull-retry-interval` is `100`. The `qfull-retries` capability is a `uchar_t` (`0` to `255`) while `qfull-retry-interval` is a `ushort_t` (`0` to `65535`).

The target driver needs to enable `tagged-qing` and `wide-xfer` explicitly. The `untagged-qing` capability is always enabled and its value cannot be modified, because `fas` can queue commands even when `tagged-qing` is disabled.

Whenever there is a conflict between the value of `scsi-options` and a capability, the value set in `scsi-options` prevails. Only `whom != 0` is supported in the `scsi_ifsetcap(9F)` call.

Refer to `scsi_ifsetcap(9F)` and `scsi_ifgetcap(9F)` for details.

Files

`/kernel/drv/fas`	ELF Kernel Module
`/kernel/drv/fas.conf`	Optional configuration file

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	Limited to Sparc SBus-based systems with FAS366-based SCSI port and SunSWIFT SBus SCSI Host Adapter/Fast Ethernet option.

See Also

`prtconf(1M)`, `driver.conf(4)`, `attributes(5)`, `scsi_abort(9F)`, `scsi_hba_attach(9F)`, `scsi_ifgetcap(9F)`, `scsi_ifsetcap(9F)`, `scsi_reset(9F)`, `scsi_sync_pkt(9F)`, `scsi_transport(9F)`, `scsi_device(9S)`, `scsi_extended_sense(9S)`, `scsi_inquiry(9S)`, `scsi_pkt(9S)`

Writing Device Drivers

ANSI Small Computer System Interface-2 (SCSI-2)

QLogic Corporation, *FAS366 Technical Manuals*.

Diagnostics

The messages described below are some that may appear on the system console, as well as being logged.

The first five messages may be displayed while the `fas` driver is trying to attach; these messages mean that the `fas` driver was unable to attach. All of these messages are preceded by "fas%d", where "%d" is the instance number of the `fas` controller.

Device in slave-only slot
The SBus device has been placed in a slave-only slot and will not be accessible; move to non-slave-only SBus slot.

Device is using a hilevel intr
The device was configured with an interrupt level that cannot be used with this `fas` driver. Check the SBus device.

Cannot alloc dma handle
Driver was unable to allocate memory for a DMA controller.

Cannot alloc cmd area
Driver was unable to allocate memory for a command address.

Cannot create kmem_cache
Driver was unable to allocate memory for internal data structures.

Unable to map FAS366 registers
Driver was unable to map device registers; check for bad hardware. Driver did not attach to device; SCSI devices will be inaccessible.

Cannot add intr
Driver could not add its interrupt service routine to the kernel.

Cannot map dma
Driver was unable to locate a DMA controller. This is an auto-configuration error.

Cannot bind cmdarea
Driver was unable to bind the DMA handle to an address.

Cannot create devctl minor node
Driver is unable to create a minor node for the controller.

Cannot attach
The driver was unable to attach; usually follows another warning that indicates why attach failed.

Disabled TQ since disconnects are disabled
Tagged queuing was disabled because disconnects were disabled in `scsi-options`.

Bad clock frequency
Check for bad hardware.

Sync of pkt (<address>) failed
Syncing a SCSI packet failed. Refer to `scsi_sync_pkt(9F)`.

All tags in use!
The driver could not allocate another tag number. The target devices do not properly support tagged queuing.

Gross error in FAS366 status
The driver experienced severe SCSI bus problems. Check cables and terminator.

Spurious interrupt
The driver received an interrupt while the hardware was not interrupting.

Lost state in phasemanage
The driver is confused about the state of the SCSI bus.

Unrecoverable DMA `error during selection`
The DMA controller experienced host SBus problems. Check for bad hardware.

Bad sequence step (<step number>) in selection
The FAS366 hardware reported a bad sequence step. Check for bad hardware.

Undetermined selection failure
The selection of a target failed unexpectedly. Check for bad hardware.

Target <n>: failed reselection (bad reselect bytes)
A reconnect failed, target sent incorrect number of message bytes. Check for bad hardware.

Target <n>: failed reselection (bad identify message)
A reconnect failed, target didn't send identify message or it got corrupted. Check for bad hardware.

Target <n>: failed reselection (not in msgin phase)
Incorrect SCSI bus phase after reconnection. Check for bad hardware.

Target <n>: failed reselection (unexpected bus free)
Incorrect SCSI bus phase after reconnection. Check for bad hardware.

Target <n>: failed reselection (timeout on receiving tag msg)
A reconnect failed; target failed to send tag bytes. Check for bad hardware.

Target <n>: failed reselection (botched tag)
A reconnect failed; target failed to send tag bytes. Check for bad hardware.

Target <n>: failed reselection (invalid tag)
A reconnect failed; target sent incorrect tag bytes. Check for bad hardware.

Target <n>: failed reselection (Parity error in reconnect msg's)
A reconnect failed; parity error detected. Check for bad hardware.

Target <n>: failed reselection (no command)
A reconnect failed; target accepted `abort` or `reset`, but still tries to reconnect. Check for bad hardware.

Unexpected bus free
Target disconnected from the bus without notice. Check for bad hardware.

Target <n> didn't disconnect after sending <message>
The target unexpectedly did not disconnect after sending <*message*> .

Bad sequence step (0x?) in selection
The sequence step register shows an improper value. The target might be misbehaving.

Illegal dma boundary?
An attempt was made to cross a boundary that the driver could not handle.

Unwanted data xfer direction for Target <n>
The target went into an unexpected phase.

Unrecoverable DMA error on dma <send/receive>
There is a DMA error while sending/receiving data. The host DMA controller is experiencing some problems.

SCSI bus DATA IN phase parity error
The driver detected parity errors on the SCSI bus.

SCSI bus MESSAGE IN phase parity error
The driver detected parity errors on the SCSI bus.

SCSI bus STATUS phase parity `error`
The driver detected parity errors on the SCSI bus.

Premature end of extended message
An extended SCSI bus message did not complete. Suspect a target firmware problem.

Premature end of input message
A multibyte input message was truncated. Suspect a target firmware problem.

Input message botch
The driver is confused about messages coming from the target.

Extended message <n> is too long
The extended message sent by the target is longer than expected.

<name> message <n> from Target <m> garbled
Target <*m*> sent message <*name*> of value <*n*> which the driver did not understand.

Target <n> rejects our message <name>
Target <*n*> rejected a message sent by the driver.

Rejecting message <name> from Target <n>
The driver rejected a message received from target <*n*>.

Cmd transmission `error`
The driver was unable to send out command bytes.

Target <n> refused message resend
The target did not accept a message resend.

MESSAGE OUT phase parity error
The driver detected parity errors on the SCSI bus.

Two byte message <name> <value> rejected
The driver does not accept this two byte message.

Gross error in fas status <stat>
The `fas` chip has indicated a gross error like FIFO overflow.

Polled cmd failed (target busy)
A polled command failed because the target did not complete outstanding commands within a reasonable time.

Polled cmd failed
A polled command failed because of timeouts or bus errors.

Auto request sense failed
Driver is unable to get request sense from the target.

Disconnected command timeout for Target <id>.<lun>
A timeout occurred while target `id`/*lun* was disconnected. This is usually a target firmware problem. For tagged queuing targets, <*n*> commands were outstanding when the timeout was detected.

Disconnected tagged cmds (<n>) timeout for Target <id>.<lun>
A timeout occurred while target `id`/*lun* was disconnected. This is usually a target firmware problem. For tagged queuing targets, <*n*> commands were outstanding when the timeout was detected.

Connected command timeout for Target <id>.<lun>
This is usually a SCSI bus problem. Check cables and termination.

Target <id>.<lun> reverting to async. mode
A data transfer hang was detected. The driver attempts to eliminate this problem by reducing the data transfer rate.

Target <id>.<lun> reducing sync. transfer rate
A data transfer hang was detected. The driver attempts to eliminate this problem by reducing the data transfer rate.

Reverting to slow SCSI cable mode
A data transfer hang was detected. The driver attempts to eliminate this problem by reducing the data transfer rate.

Target <id> reducing sync. transfer rate
A data transfer hang was detected. The driver attempts to eliminate this problem by reducing the data transfer rate.

Target <id> reverting to async. mode
A data transfer hang was detected. The driver attempts to eliminate this problem by reducing the data transfer rate.

Target <id> disabled wide SCSI mode
Due to problems on the SCSI bus, the driver goes into more conservative mode of operation to avoid further problems.

Reset SCSI bus failed
An attempt to reset the SCSI bus failed.

External SCSI bus reset
Another initiator reset the SCSI bus.

Warnings The `fas` hardware (FAS366) supports both Wide and Fast SCSI mode, but fast20 is not supported. The maximum SCSI bandwidth is 20 MB/sec. Initiator mode block sequence (IBS) is not supported.

Notes The `fas` driver exports properties indicating per target the negotiated transfer speed (`target<n>-sync-speed`), whether wide bus is supported (`target<n>-wide`), `scsi-options` for that particular target (`target<n>-scsi-options`), and whether tagged queuing has been enabled (`target<n>-TQ`). The `sync-speed` property value is the data transfer rate in KB/sec. The `target<n>-TQ` and the `target<n>-wide` property have value `1` to indicate that the corresponding capability is enabled, or `0` to indicate that the capability is disabled for that target. Refer to `prtconf(1M)` (verbose option) for viewing the `fas` properties.

```
SUNW,fas,instance #1
 Driver software properties:
     name <target3-TQ> length <4>
         value <0x00000001>.
     name <target3-wide> length <4>
         value <0x00000000>.
     name <target3-sync-speed> length <4>
         value <0x00002710>.
     name <target3-scsi-options> length <4>
         value <0x000003f8>.
     name <target0-TQ> length <4>
         value <0x00000001>.
     name <pm_norm_pwr> length <4>
         value <0x00000001>.
     name <pm_timestamp> length <4>
         value <0x30040346>.
     name <scsi-options> length <4>
         value <0x000003f8>.
     name <scsi-watchdog-tick> length <4>
         value <0x0000000a>.
     name <scsi-tag-age-limit> length <4>
         value <0x00000002>.
     name <scsi-reset-delay> length <4>
         value <0x00000bb8>.
 Register Specifications:
     Bus Type=0x3, Address=0x8800000, Size=10
     Bus Type=0x3, Address=0x8810000, Size=40
 Interrupt Specifications:
     Interrupt Priority=0x35 (ipl 5)
```

Name fasttrap – DTrace user instruction tracing provider

Description The `fasttrap` driver is a DTrace dynamic tracing provider that performs dynamic instrumentation of arbitrary instructions in Solaris processes. The `fasttrap` driver implements the DTrace `fasttrap` and `pid` providers.

The `fasttrap` driver is not a public interface and you access instrumentation offered by this provider through DTrace. Refer to the *Solaris Dynamic Tracing Guide* for a description of the public documented interfaces available for the DTrace facility and the probes offered by the `fasttrap` provider.

Sparc Only The `fasttrap` provider provides a DTrace probe that fires each time a user process executes an instruction. The `pid` provider allows for the dynamic creation of DTrace probes corresponding to instruction locations inside any user process specified using a process ID and an instruction address or symbol name. Together these providers permit DTrace users to perform instrumentation of Solaris user processes and to trace the interactions between processes and the operating system. See the chapter entitled “User Process Tracing" in the *Solaris Dynamic Tracing Guide* for information on how to use these providers to instrument processes.

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWdtrp
Interface Stability	Private

See Also `dtrace(1M)`, `attributes(5)`, `dtrace(7D)`

Solaris Dynamic Tracing Guide

Name fbio – frame buffer control operations

Description The frame buffers provided with this release support the same general interface that is defined by <sys/fbio.h>. Each responds to an FBIOGTYPE ioctl(2) request which returns information in a fbtype structure.

Each device has an FBTYPE which is used by higher-level software to determine how to perform graphics functions. Each device is used by opening it, doing an FBIOGTYPE ioctl() to see which frame buffer type is present, and thereby selecting the appropriate device-management routines.

FBIOGINFO returns information specific to the GS accelerator.

FBIOSVIDEO and FBIOGVIDEO are general-purpose ioctl() requests for controlling possible video features of frame buffers. These ioctl() requests either set or return the value of a flags integer. At this point, only the FBVIDEO_ON option is available, controlled by FBIOSVIDEO. FBIOGVIDEO returns the current video state.

The FBIOSATTR and FBIOGATTR ioctl() requests allow access to special features of newer frame buffers. They use the fbsattr and fbgattr structures.

Some color frame buffers support the FBIOPUTCMAP and FBIOGETCMAP ioctl() requests, which provide access to the colormap. They use the fbcmap structure.

Also, some framebuffers with multiple colormaps will either encode the colormap identifier in the high-order bits of the index field in the fbcmap structure, or use the FBIOPUTCMAPI and FBIOGETCMAPI ioctl() requests.

FBIOVERTICAL is used to wait for the start of the next vertical retrace period.

FBIOVRTOFFSET Returns the offset to a read-only *vertical retrace page* for those framebuffers that support it. This vertical retrace page may be mapped into user space with mmap(2). The first word of the vertical retrace page (type unsigned int) is a counter that is incremented every time there is a vertical retrace. The user process can use this counter in a variety of ways.

FBIOMONINFO returns a mon_info structure which contains information about the monitor attached to the framebuffer, if available.

FBIOSCURSOR, FBIOGCURSOR, FBIOSCURPOS and FBIOGCURPOS are used to control the hardware cursor for those framebuffers that have this feature. FBIOGCURMAX returns the maximum sized cursor supported by the framebuffer. Attempts to create a cursor larger than this will fail.

Finally FBIOSDEVINFO and FBIOGDEVINFO are used to transfer variable-length, device-specific information into and out of framebuffers.

See Also ioctl(2), mmap(2)

Bugs The FBIOSATTR and FBIOGATTR ioctl() requests are only supported by frame buffers which emulate older frame buffer types. If a frame buffer emulates another frame buffer, FBIOGTYPE returns the emulated type. To get the real type, use FBIOGATTR.

The FBIOGCURPOS ioctl was incorrectly defined in previous operating systems, and older code running in binary compatibility mode may get incorrect results.

Name fbt – DTrace function boundary tracing provider

Description The `fbt` driver is a DTrace dynamic tracing provider that performs dynamic instrumentation at function boundaries in the Solaris kernel.

The function is the fundamental unit of program text. In a well-designed system, the function performs a discrete and well-defined operation on a specified object or series of like objects. Most functions are implemented by themselves calling functions on encapsulated objects, but some functions —so-called "leaf functions" — are implemented without making further function calls. The Function Boundary Tracing `fbt` provider contains a mechanism for instrumenting the vast majority of functions in the kernel and offering the instrumentation as a set of DTrace probes.

The `fbt` driver is not a public interface and you access the instrumentation offered by this provider through DTrace. Refer to the *Solaris Dynamic Tracing Guide* for a description of the public documented interfaces available for the DTrace facility and the probes offered by the `fbt` provider.

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWdtrp
Interface Stability	Private

See Also `dtrace(1M)`, `attributes(5)`, `dtrace(7D)`

Solaris Dynamic Tracing Guide

Name fcip – IP/ARP over Fibre Channel datagram encapsulation driver

Synopsis `/dev/fcip`

Description The `fcip` driver is a Fibre Channel upper layer protocol module for encapsulating IP (IPv4) and ARP datagrams over Fibre Channel. The `fcip` driver is a loadable, clonable, STREAMS driver supporting the connectionless Data Link Provider Interface, `dlpi(7P)` over any Sun Fibre Channel transport layer-compliant host adapter.

The `fcip` driver complies with the *RFC 2625* specification for encapsulating IP/ARP datagrams over Fibre Channel, and allows encapsulation of IPv4 only, as specified in *RFC 2625*. The `fcip` driver interfaces with the `fp(7d)` Sun Fibre Channel port driver.

Application Programming Interface The cloning character-special device `/dev/fcip` is used to access all Fibre Channel ports capable of supporting IP/ARP traffic on the system.

fcip and DLPI The `fcip` driver is a "style 2" Data Link Service Provider. All `M_PROTO` and `M_PCPROTO` type messages are interpreted as DLPI primitives. Valid DLPI primitives are defined in `<sys/dlpi.h>`. Refer to `dlpi(7P)` for more information on DLPI primitives.

An explicit `DL_ATTACH_REQ` message must be sent to associate the opened stream with a particular Fibre Channel port (ppa). The ppa ID is interpreted as an unsigned long data type and indicates the corresponding Fibre Channel port driver instance number. An error (`DL_ERROR_ACK`) is returned by the driver if the ppa field value does not correspond to a valid port driver instance number or if the Fibre Channel port is not `ONLINE`. Refer to `fp(7d)` for more details on the Fibre Channel port driver.

The values returned by the driver in the `DL_INFO_ACK` primitive in response to a `DL_INFO_REQ` from the user are as follows:

- Maximum SDU is `65280` (defined in *RFC 2625*).
- Minimum SDU is `0`.
- DLSAP address length is `8`.
- MAC type is `DL_ETHER`.
- SAP length is `-2`.
- Service mode is `DL_CLDLS`.
- Optional quality of service (QOS) fields are set to `0`.
- Provider style is `DL_STYLE2`.
- Provider version is `DL_VERSION_2`.
- Broadcast address value is `0xFFFFFFFF`.

Once in `DL_ATTACHED` state, the user must send a `DL_BIND_REQ` to associate a particular SAP (Service Access Point) with the stream. The `fcip` driver DLSAP address format consists of the 6–byte physical address component followed immediately by the 2–byte SAP component producing an 8–byte DLSAP address. Applications should not be programmed to use this implementation-specific DLSAP address format, but use information returned in the `DL_INFO_ACK` primitive to compose and decompose DLSAP addresses. The SAP length, full

DLSAP length, and SAP/physical ordering are included within the DL_INFO_ACK. The physical address length is the full DLSAP address length minus the SAP length. The physical address length can also be computed by issuing the DL_PHYS_ADDR_REQ primitive to obtain the current physical address associated with the stream.

Once in the DL_BOUND state, the user can transmit frames on the fibre by sending DL_UNITDATA_REQ messages to the fcip driver. The fcip driver will route received frames up any of the open and bound streams having a SAP which matches the received frame's SAP type as DL_UNITDATA_IND messages. Received Fibre Channel frames are duplicated and routed up multiple open streams if necessary. The DLSAP address contained within the DL_UNITDATA_REQ and DL_UNITDATA_IND messages consists of both the SAP (type) and physical address (WorldWideName) components.

Other Primitives In Fibre Channel, *multicasting* is defined as an optional service for Fibre Channel classes three and six only. If required, the Fibre Channel broadcast service can be used for multicasting. The *RFC 2625* specification does not support IP multicasting or promiscuous mode.

fcip Fibre Channel ELS The fcip driver will use the FARP Fibre Channel Extended Link Service (ELS), where supported, to resolve WorldWide Names (MAC address) to FC Port Identifiers(Port_ID). The fcip driver also supports InARP to resolve WorldWide Name and Port_ID to an IP address.

Files

/dev/fcip	fcip character-special device
/kernel/drv/fcip	32–bit ELF kernel driver (x86)
/kernel/drv/amd64/fcip	64–bit ELF kernel driver (x86)
/kernel/drv/sparcv9/fcip	64–bit ELF kernel driver (SPARC)
/kernel/drv/fcip.conf	fcip driver configuration file

Attributes

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC
Availability	SUNWfcip

See Also

netstat(1M), prtconf(1M), driver.conf(4), fp(7d), dlpi(7P)

Writing Device Drivers

IP and ARP over Fibre Channel, RFC 2625 M. Rajagopal, R. Bhagwat, W. Rickard. Gadzoox Networks, June 1999

ANSI X3.230-1994, Fibre Channel Physical and Signalling Interface (FC-PH)

ANSI X3.272-1996, Fibre Channel Arbitrated Loop (FC-AL)

Notes If you use a Fibre Channel adapter with two or more ports that each share a common Node WorldWideName, the `fcip` driver will likely attach to the first port on the adapter.

RFC 2625 requires that both source and destination WorldWideNames have their 4 bit NAA identifiers set to binary '0001,' indicating that an IEEE 48–bit MAC address is contained in the lower 48 bits of the network address fields. For additional details, see the *RFC 2625* specification.

Name fcoe – fibre channel over Ethernet transport driver

Description The `fcoe` driver is a pseudo nexus driver which supports the transportation of FCoE encapsualted frames. FCoE Ethernet frame will encapsulate the raw Fibre Channel frame.

The `fcoe` driver interfaces with FCoE target mode device driver, `fcoet(7D)`.

Files

`/kernel/drv/fcoe`	32-bit ELF kernel module (x86)
`/kernel/drv/amd64/fcoe`	64-bit ELF kernel module (x86)
`kernel/drv/sparcv`	64-bit ELF kernel module (SPARC)

Attributes See `attributes(5)` for a description of the following attribute:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86
Availability	SUNWfcoe

See Also `driver.conf(4)`, `attributes(5)`, `fcoet(7D)`

Writing Device Drivers

ANSI X3.269-1996, Fibre Channel Protocol for SCSI (FCP)

Name fcoei – Fibre Channel Over Ethernet Initiator Mode Driver

Synopsis `fcoei@port,0`

Description The `fcoei` driver is a pseudo device driver which encapsulates the raw Fibre Channel frames into FCoE ethernet frames, or decapsulates FC frames from FCoE ethernet frames. The supported FC frames include extended/basic link services, common transport frames and initiator mode FCP frames.

The `fcoei` driver interfaces with the Sun Fibre Channel port driver, `fp(7d)`, and the FCoE transport driver, `fcoe(7D)`.

Files

`/kernel/drv/fcoei`	32-bit ELF kernel module (x86)
`/kernel/drv/amd64/fcoei`	64-bit ELF kernel module (x86)
`kernel/drv/sparcv/fcoei`	64-bit ELF kernel module (SPARC)

Attributes See `attributes(5)` for a description of the following attribute:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86
Availability	SUNWfcoei

See Also `driver.conf(4)`, `attributes(5)`, `fcoe(7D)`, `fcoet(7D)`, `fp(7d)`

Writing Device Drivers

ANSI X3.269-1996, Fibre Channel Protocol for SCSI (FCP)

Name fcoet – fibre channel over Ethernet target mode driver

Description The `fcoet` driver is a pseudo device driver which encapsulates the raw Fibre Channel frames into FCoE Ethernet frames, or decapsulates FC frames from FCoE Ethernet frames. The supported FC frames contain extended/basic link services, common transport frames and target mode FCP frames.

The `fcoet` driver interfaces with COMSTAR FC transport driver, `fct`, and FCoE transport driver, `fcoe`(7D).

Files

`/kernel/drv/fcoet`	32-bit ELF kernel module (x86)
`/kernel/drv/amd64/fcoet`	64-bit ELF kernel module (x86)
`/kernel/drv/sparcv9/fcoet`	64-bit ELF kernel module (SPARC)

Attributes See `attributes`(5) for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86
Availability	SUNWfcoet

See Also `driver.conf`(4), `attributes`(5), `fcoe`(7D)

Writing Device Drivers

ANSI X3.269-1996, Fibre Channel Protocol for SCSI (FCP)

Name fcp – Fibre Channel protocol driver

Description The `fcp` driver is the upper layer protocol that supports mechanisms for transporting SCSI-3 commands over Fibre Channel. The `fcp` driver, which interfaces with the Sun Fibre Channel transport library `fctl(7D)`, supports the standard functions provided by the SCSA interface.

Files

`/kernel/drv/fcp`	32–bit ELF kernel driver (x86)
`/kernel/drv/amd64/fcp`	64–bit ELF kernel driver (x86)
`/kernel/drv/sparcv9/fcp`	64–bit ELF kernel driver (SPARC)

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC
Interface stability	Unknown
Availability	SUNWfcp

See Also `prtconf(1M)`, `driver.conf(4)`, `fctl(7D)`, `fp(7d)`, `usoc(7D)`

Writing Device Drivers

Fibre Channel Physical and Signaling Interface (FC-PH) ANSI X3.230: 1994

Fibre Channel Generic Services (FC-GS-2) Project 1134-D

Fibre Channel Arbitrated Loop (FC-AL) ANSI X3.272-1996

Fibre Channel Protocol for SCSI (FCP) ANSI X3.269-1996

SCSI-3 Architecture Model (SAM) Fibre Channel Private Loop SCSI Direct Attach (FC-PLDA) ANSI X3.270-1996

Fabric Loop Attachment (FC-FLA), NCITS TR-20:1998

Name fctl – Sun Fibre Channel transport library

Description The `fctl` kernel module interfaces the Sun Fibre Channel upper layer protocol (ULP) mapping modules with Sun Fibre Channel adapter (FCA) drivers. There are no user-configurable options for this module.

Files

`/kernel/misc/fctl`	32–bit ELF kernel module (x86)
`/kernel/misc/amd64/fctl`	64–bit ELF kernel module (x86)
`/kernel/misc/sparcv9/fctl`	64–bit ELF kernel module (SPARC)

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC
Interface stability	Unknown
Availability	SUNWfctl

See Also `fp(7d)`

Name fd, fdc – drivers for floppy disks and floppy disk controllers

Synopsis

SPARC `/dev/diskette0`

`/dev/rdiskette0`

x86 `/dev/diskette[0-1]`

`/dev/rdiskette[0-1]`

Description The `fd` and `fdc` drivers provide the interfaces to floppy disks using the Intel 8272, Intel 82077, NEC 765, or compatible disk controllers on x86 based systems.

The default partitions for the floppy driver are:

a All cylinders except the last

b Only the last cylinder

c Entire diskette

The `fd` driver autosenses the density of the diskette.

When the floppy is first opened the driver looks for a SunOS label in logical block 0 of the diskette. If attempts to read the SunOS label fail, the open will fail. If block 0 is read successfully but a SunOS label is not found, auto-sensed geometry and default partitioning are assumed.

The `fd` driver supports both block and raw interfaces.

The block files (`/dev/diskette*`) access the diskette using the system's normal buffering mechanism and may be read and written without regard to physical diskette records.

There is also a raw (`/dev/rdiskette*`) interface that provides for direct transmission between the diskette and the user's read or write buffer. A single `read(2)` or `write(2)` call usually results in one I/O operation; therefore raw I/O is considerably more efficient when larger blocking factors are used. A blocking factor of no less than 8 Kbytes is recommended. See the Notes section, below, for information on the number of sectors per track.

3.5" Diskettes For 3.5" double-sided diskettes, the following densities are supported:

SPARC	1.7 Mbyte density	80 cylinders, 21 sectors per track, 1.7 Mbyte capacity
	high density	80 cylinders, 18 sectors per track, 1.44 Mbyte capacity
	double density	80 cylinders, 9 sectors per track, 720 Kbyte capacity
x86	extended density	80 cylinders, 36 sectors per track, 2.88 Mbyte capacity
	1.7 Mbyte density	80 cylinders, 21 sectors per track, 1.7 Mbyte capacity

	high density	80 cylinders, 18 sectors per track, 1.44 Mbyte capacity
	double density	80 cylinders, 9 sectors per track, 760 Kbyte capacity

5.25" Diskettes For 5.25" double-sided diskettes on x86 platforms, the densities listed below are supported:

SPARC 5.25" diskettes are not supported on SPARC platforms.

x86	high density	80 cylinders, 15 sectors per track, 1.2 Mbyte capacity
	double density	40 cylinders, 9 sectors per track, 360 Kbyte capacity
	double density	40 cylinders, 8 sectors per track, 320 Kbyte capacity
	quad density	80 cylinders, 9 sectors per track, 720 Kbyte capacity
	double density	40 cylinders, 16 sectors per track (256 bytes per sector), 320 Kbyte capacity
	double density	40 cylinders, 4 sectors per track (1024 bytes per sector), 320 Kbyte capacity

Errors

`EBUSY`	During opening, the partition has been opened for exclusive access and another process wants to open the partition. Once open, this error is returned if the floppy disk driver attempted to pass a command to the floppy disk controller when the controller was busy handling another command. In this case, the application should try the operation again.
`EFAULT`	An invalid address was specified in an ioctl command (see `fdio(7I)`).
`EINVAL`	The number of bytes read or written is not a multiple of the diskette's sector size. This error is also returned when an unsupported command is specified using the `FDIOCMD` ioctl command (see `fdio(7I)`).
`EIO`	During opening, the diskette does not have a label or there is no diskette in the drive. Once open, this error is returned if the requested I/O transfer could not be completed.
`ENOSPC`	An attempt was made to write past the end of the diskette.
`ENOTTY`	The floppy disk driver does not support the requested ioctl functions (see `fdio(7I)`).
`ENXIO`	The floppy disk device does not exist or the device is not ready.
`EROFS`	The floppy disk device is opened for write access and the diskette in the drive is write protected.
x86 Only `ENOSYS`	The floppy disk device does not support the requested ioctl function (`FDEJECT`).

x86 Configuration The driver attempts to initialize itself using the information found in the configuration file, `/platform/i86pc/kernel/drv/fd.conf`.

```
name="fd"  parent="fdc"  unit=0;
name="fd"  parent="fdc"  unit=1;
```

Files

SPARC	`/platform/sun4u/kernel/drv/fd`	driver module
	`/usr/include/sys/fdreg.h`	structs and definitions for Intel 82072 and 82077 controllers
	`/usr/include/sys/fdvar.h`	structs and definitions for floppy drivers
	`/dev/diskette`	device file
	`/dev/diskette0`	device file
	`/dev/rdiskette`	raw device file
	`/dev/rdiskette0`	raw device file

For ucb Compatibility

`/dev/fd0[a-c]`	block file
`/dev/rfd0[a-c]`	raw file
`/dev/diskette0`	directory containing volume management character device file
`/dev/rdiskette0`	directory containing the volume management raw character device file
`/dev/aliases/floppy0`	symbolic link to the entry in `/dev/rdiskette0`

x86	`/platform/i86pc/kernel/drv/fd`	driver module
	`/platform/i86pc/kernel/drv/fd.conf`	configuration file for floppy driver
	`/platform/i86pc/kernel/drv/fdc`	floppy-controller driver module
	`/platform/i86pc/kernel/drv/fdc.conf`	configuration file for the floppy-controller
	`/usr/include/sys/fdc.h`	structs and definitions for x86 floppy devices
	`/usr/include/sys/fdmedia.h`	structs and definitions for x86 floppy media

x86 First Drive

`/dev/diskette`	device file
`/dev/diskette0`	device file
`/dev/rdiskette`	raw device file
`/dev/rdiskette0`	raw device file

For ucb Compatibility

`/dev/fd0[a-c]`	block file

`/dev/rfd0[a-c]`	raw file
`/dev/diskette0`	directory containing volume management character device file
`/dev/rdiskette0`	directory containing the volume management raw character device file
`/dev/aliases/floppy0`	symbolic link to the entry in /dev/rdiskette0

x86 Second Drive

`/dev/diskette1`	device file
`/dev/rdiskette1`	raw device file

For ucb Compatibility

`/dev/fd1[a-c]`	block file
`/dev/rfd1[a-c]`	raw file
`/dev/diskette1`	directory containing volume management character device file
`/dev/rdiskette1`	directory containing the volume management raw character device file
`/dev/aliases/floppy1`	symbolic link to the entry in `/dev/rdiskette1`

See Also `fdformat(1)`, `dd(1M)`, `drvconfig(1M)`, `read(2)`, `write(2)`, `driver.conf(4)`, `dkio(7I)` `fdio(7I)`

Diagnostics

All Platforms `fd<n>: <command name> failed (<sr1> <sr2> <sr3>)`

The <command name> failed after several retries on drive <n>. The three hex values in parenthesis are the contents of status register 0, status register 1, and status register 2 of the Intel 8272, the Intel 82072, and the Intel 82077 Floppy Disk Controller on completion of the command, as documented in the data sheet for that part. This error message is usually followed by one of the following, interpreting the bits of the status register:

`fd<n>:` `not writable`

`fd<n>:` `crc error blk <block number>`

There was a data error on <block number>.

`fd<n>:` `bad format`

`fd<n>:` `timeout`

`fd<n>:` `drive not ready`

`fd<n>:` `unformatted diskette or no diskette in drive`

`fd<n>:` `block <block number> is past the end!`

`(nblk=<total number of blocks>)`

The operation tried to access a block number that is greater than the total number of blocks.

`fd<n>:` `b_bcount 0x<op_size> not % 0x<sect_size>`

The size of an operation is not a multiple of the sector size.

`fd<n>:` `overrun/underrun`

`fd<n>:` `host bus error`. There was a hardware error on a system bus.

SPARC Only Overrun/underrun errors occur when accessing a diskette while the system is heavily loaded. Decrease the load on the system and retry the diskette access.

Notes 3.5" high density diskettes have 18 sectors per track and 5.25" high density diskettes have 15 sectors per track. They can cross a track (though not a cylinder) boundary without losing data, so when using `dd(1M)` or `read(2)`/`write(2)` calls to or from the raw diskette, you should specify `bs=18k` or multiples thereof for 3.5" diskettes, and `bs=15k` or multiples thereof for 5.25" diskettes.

The SPARC `fd` driver is *not* an unloadable module.

Under Solaris (x86 Edition), the configuration of the floppy drives is specified in CMOS configuration memory. Use the BIOS setup program for the system to define the diskette size and density/capacity for each installed drive. Note that MS-DOS may operate the floppy drives correctly, even though the CMOS configuration may be in error. Solaris (x86 Edition) relies on the CMOS configuration to be accurate.

Name fdio – floppy disk control operations

Synopsis

```
#include <sys/fdio.h>
```

Description The Solaris floppy driver supports a set of ioctl(2) requests for getting and setting the floppy drive characteristics. Basic to these `ioctl( )` requests are the definitions in `<sys/fdio.h>`.

ioctls The following `ioctl( )` requests are available on the Solaris floppy driver.

`FDDEFGEOCHAR` x86 based systems: This `ioctl( )` forces the floppy driver to restore the diskette and drive characteristics and geometry, and partition information to default values based on the device configuration.

`FDGETCHANGE` The argument is a pointer to an `int`. This `ioctl( )` returns the status of the diskette-changed signal from the floppy interface. The following defines are provided for cohesion.

Note: For x86 based systems, use `FDGC_DETECTED` (which is available only on x86 based systems) instead of `FDGC_HISTORY`.

```
/*
 * Used by FDGETCHANGE, returned state of the sense disk change bit.
 */
#define FDGC_HISTORY  0x01     /* disk has changed since insertion or
                                  last FDGETCHANGE call */
#define FDGC_CURRENT  0x02     /* if set, indicates drive has floppy,
>                                 otherwise, drive is empty */
#define FDGC_CURWPROT 0x10     /* current state of write protect */
#define FDGC_DETECTED 0x20     /* previous state of DISK CHANGE */
```

`FDIOGCHAR` The argument is a pointer to an `fd_char` structure (described below). This `ioctl( )` gets the characteristics of the floppy diskette from the floppy controller.

`FDIOSCHAR` The argument is a pointer to an `fd_char` structure (described below). This `ioctl( )` sets the characteristics of the floppy diskette for the floppy controller. Typical values in the `fd_char` structure for a high density diskette:

```
field value
fdc_medium    0
fdc_transfer_rate    500
fdc_ncyl    80
fdc_nhead    2
fdc_sec_size    512
fdc_secptrack    18
fdc_steps    -1    { This field doesn't apply. }
```

```
/*
 * Floppy characteristics
 */
```

```
struct fd_char {
 uchar_t fdc_medium;     /* equals 1 if floppy is medium density format */
 int fdc_transfer_rate;  /* transfer rate */
 int fdc_ncyl;           /* number of cylinders */
 int fdc_nhead;          /* number of heads */
 int fdc_sec_size;       /* sector size */
 int fdc_secptrack;      /* sectors per track */
 int fdc_steps;          /* no. of steps per data track */
};
```

FDGETDRIVECHAR The argument to this `ioctl( )` is a pointer to an `fd_drive` structure (described below). This `ioctl( )` gets the characteristics of the floppy drive from the floppy controller.

FDSETDRIVECHAR x86 based systems: The argument to this `ioctl( )` is a pointer to an `fd_drive` structure (described below). This `ioctl( )` sets the characteristics of the floppy drive for the floppy controller. Only `fdd_steprate`, `fdd_headsettle`, `fdd_motoron`, and `fdd_motoroff` are actually used by the floppy disk driver.

```
/*
 * Floppy Drive characteristics
 */
struct fd_drive {
    int    fdd_ejectable;    /* does the drive support eject? */
    int    fdd_maxsearch;    /* size of per-unit search table */
    int    fdd_writeprecomp; /* cyl to start write precompensation */
    int    fdd_writereduce;  /* cyl to start recucing write current */
    int    fdd_stepwidth;    /* width of step pulse in 1 us units */
    int    fdd_steprate;     /* step rate in 100 us units */
    int    fdd_headsettle;   /* delay, in 100 us units */
    int    fdd_headload;     /* delay, in 100 us units */
    int    fdd_headunload;   /* delay, in 100 us units */
    int    fdd_motoron;      /* delay, in 100 ms units */
    int    fdd_motoroff;     /* delay, in 100 ms units */
    int    fdd_precomplevel; /* bit shift, in nano-secs */
    int    fdd_pins;         /* defines meaning of pin 1, 2, 4 and 34 */
    int    fdd_flags;        /* TRUE READY, Starting Sector #, & Motor On */
};
```

FDGETSEARCH Not available.

FDSETSEARCH Not available.

FDEJECT SPARC: This `ioctl( )` requests the floppy drive to eject the diskette.

FDIOCMD The argument is a pointer to an `fd_cmd` structure (described below). This `ioctl( )` allows access to the floppy diskette using the floppy device driver. Only the `FDCMD_WRITE`, `FDCMD_READ`, and `FDCMD_FORMAT_TRACK` commands

are currently available.

```
struct fd_cmd {
    ushort_t fdc_cmd;      /* command to be executed */
    int      fdc_flags;    /* execution flags (x86 only) */
    daddr_t  fdc_blkno;    /* disk address for command */
    int      fdc_secnt;    /* sector count for command */
    caddr_t  fdc_bufaddr;  /* user's buffer address */
    uint_t   fdc_buflen;   /* size of user's buffer */
};
```

Please note that the `fdc_buflen` field is currently unused. The `fdc_secnt` field is used to calculate the transfer size, and the buffer is assumed to be large enough to accommodate the transfer.

```
{
/*
* Floppy commands
 */
#define    FDCMD_WRITE    1
#define    FDCMD_READ    2
#define    FDCMD_SEEK    3
#define    FDCMD_REZERO    4
#define    FDCMD_FORMAT_UNIT    5
#define    FDCMD_FORMAT_TRACK    6
};
```

FDRAW The argument is a pointer to an `fd_raw` structure (described below). This `ioctl( )` allows direct control of the floppy drive using the floppy controller. Refer to the appropriate floppy-controller data sheet for full details on required command bytes and returned result bytes. The following commands are supported.

```
/*
* Floppy raw commands
*/
#define FDRAW_SPECIFY    0x03
#define FDRAW_READID    0x0a    (x86 only)
#define FDRAW_SENSE_DRV    0x04
#define FDRAW_REZERO    0x07
#define FDRAW_SEEK    0x0f
#define FDRAW_SENSE_INT    0x08    (x86 only)
#define FDRAW_FORMAT    0x0d
#define FDRAW_READTRACK    0x02
#define FDRAW_WRCMD    0x05
#define FDRAW_RDCMD    0x06
#define FDRAW_WRITEDEL    0x09
#define FDRAW_READDEL   0x0c
```

Please note that when using FDRAW_SEEK or FDRAW_REZERO, the driver automatically issues a FDRAW_SENSE_INT command to clear the interrupt from the FDRAW_SEEK or the FDRAW_REZERO. The result bytes returned by these commands are the results from the FDRAW_SENSE_INT command. Please see the floppy-controller data sheet for more details on FDRAW_SENSE_INT.

```
/*
 * Used by FDRAW
 */
struct    fd_raw {
 char     fdr_cmd[10];     /* user-supplied command bytes */
 short    fdr_cnum;        /* number of command bytes */
 char     fdr_result[10];  /* controller-supplied result bytes */
 ushort_t fdr_nbytes;      /* number to transfer if read/write command */
 char     *fdr_addr;       /* where to transfer if read/write command */
};
```

See Also ioctl(2), dkio(7I), fd(7D), hdio(7I)

Name ffb – 24-bit UPA color frame buffer and graphics accelerator

Description `ffb` is a 24-bit UPA-based color frame buffer and graphics accelerator which comes in the two configurations: single buffered frame and double buffered frame.

Single buffered frame buffer	Consists of 32 video memory planes of 1280 x 1024 pixels, including 24-bit single-buffering and 8-bit X planes.
Double buffered frame buffer	Consists of 96 video memory planes of 1280 x 1024 pixels, including 24-bit double-buffering, 8-bit X planes, 28-bit Z-buffer planes and 4-bit Y planes.

The driver supports the following frame buffer ioctls which are defined in `fbio(7I)`:

```
FBIOPUTCMAP, FBIOGETCMAP, FBIOSVIDEO, FBIOGVIDEO, FBIOVERTICAL,
FBIOSCURSOR, FBIOGCURSOR, FBIOSCURPOS, FBIOGCURPOS,  FBIOGCURMAX,
FBIO_WID_PUT,  FBIO_WID_GET
```

Files `/dev/fbs/ffb0` device special file

See Also `ffbconfig(1M)`, `mmap(2)`, `fbio(7I)`

Name flowacct – Flow Accouting module

Description The flow accounting module `flowacct` enables you to record flow details. You use flow details to gather statistics and/or for billing purposes. Accounting consists of recording flow details in a location you designate and in a format that you can retrieve at a later stage. IPQoS accounting relies on the exacct mechanism to store and retrieve flow information.

A flow is defined by the 5-tuple - saddr, sport, daddr, dport and protocol.

Typically, the accounting module is the last datapath element in a sequence of actions. Flow attributes include ToS/DS, user id, project id, creation time (time the flow was created), last seen (when pkts for the flow were last seen), action name (instance that recorded the flow information), nbytes and npackets. Attributes are split into groups entitled *basic* and *extended*. The basic group records only the nbytes, npackets and action name, while the extended group is a superset of the basic group and records all attributes. The attributes to be recorded, in addition to the accounting file that contains flow details, are selected using `acctadm(1M)`. The `flowacct` module does not provide a mechanism to retrieve flow information from the accounting file nor to interpret the retrieved information.

Statistics The flowacct module exports the following statistics available through kstat:

```
module: flowacct                instance: <action id>
  name: Flowacct statistics     class <action name>
        bytes_in_tbl            <bytes in the flow table>
        epackets                <packets in error>
        flows_in_tbl            <flow records in the flow table>
        nbytes                  <number of bytes through this instance>
        npackets                <number of packets>
        usedmem                 <memory, in bytes, used by the flow table>
```

Files `/kernel/ipp/sparcv9/flowacct` 64-bit module (SPARC only.)

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWqos

See Also `ipqosconf(1M)`, `acctadm(1M)`, `libexacct3LIB`, `dlcosmk(7ipp)`, `dscpmk(7ipp)`, `ipqos(7ipp)`, `ipgpc(7ipp)`, `tokenmt(7ipp)`, `tswtclmt(7ipp)`

Name fp – Sun Fibre Channel port driver

Description The `fp` driver is a Sun fibre channel nexus driver that enables fibre channel topology discovery, device discovery, fibre channel adapter port management and other capabilities through well-defined fibre channel adapter driver interfaces.

The `fp` driver requires the presence of a fabric name server in fabric and public loop topologies to discover fibre channel devices. In private loop topologies, the driver discovers devices by performing PLOGI to all valid `AL_PA`s, provided that devices do not participate in LIRP and LILP stages of loop initialization. The `fp` driver also discovers devices in N_Port point-to-point topologies.

Configuration The `fp` driver is configured by defining properties in the `fp.conf` file. Note that you must reboot the system to have any changes you make to `fp.conf` take effect. The `fp` driver supports the following properties:

mpxio-disable
: Solaris I/O multipathing is enabled or disabled on fibre channel devices with the mpxio-disable property. Specifying mpxio-disable="no" activates I/O multipathing, while mpxio-disable="yes" disables the feature. Solaris I/O multipathing may be enabled or disabled on a per port basis. Per port settings override the global setting for the specified ports. The following example shows how to disable multipathing on port 0 whose parent is /pci@8,600000/SUNW,qlc@4:

```
name="fp"    parent="/pci@8,600000/SUNW,qlc@4"    port=0
mpxio-disable="yes";
```

manual_configuration_only
: Automatic configuration of SCSI devices in the fabric is enabled by default and thus allows all devices discovered in the SAN zone to be enumerated in the kernel's device tree automatically. The `manual_configuration_only` property may be configured to disable the default behavior and force the manual configuration of the devices in the SAN. Specifying manual_configuration_only=1 disables the automatic configuration of devices.

pwwn-lun-blacklist
: Allows you to specify target port WWNs and LUN numbers you do not want configured. LUN numbers are interpreted as decimals. White spaces and commas (',') can be used in the list of LUN numbers.

```
#
# pwwn-lun-blacklist=
# "target-port-wwn,lun-list"
#
# To prevent LUNs 1 and 2 from being configured for target
# port 510000f010fd92a1 and target port 510000e012079df1, set:
#
# pwwn-lun-blacklist=
# "510000f010fd92a1,1,2",
# "510000e012079df1,1,2";
```

```
#
```

Files

`/kernel/drv/fp`	32-bit ELF kernel driver (x86)
`/kernel/drv/amd64/fp`	64-bit ELF kernel driver (x86)
`/kernel/drv/sparcv9/fp`	64–bit ELF kernel driver (SPARC)
`/kernel/drv/fp.conf`	`fp` driver configuration file.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
mpxio-disable	Uncommitted
manual_configuration_only	Obsolete
Availability	SUNWfctl

See Also `cfgadm_fp(1M)`, `prtconf(1M)`, `stmsboot(1M)`, `driver.conf(4)`, `attributes(5)`, `fcp(7D)`, `fctl(7D)`, `scsi_vhci(7D)`

Writing Device Drivers

Fibre Channel Physical and Signaling Interface (FC-PH) ANSI X3.230: 1994

Fibre Channel Generic Services (FC-GS-2) Project 1134-D

Fibre Channel Arbitrated Loop (FC-AL) ANSI X3.272-1996

Fibre Channel Protocol for SCSI (FCP) ANSI X3.269-1996

SCSI-3 Architecture Model (SAM) Fibre Channel Private Loop SCSI Direct Attach (FC-PLDA) ANSI X3.270-1996

SCSI Direct Attach (FC-PLDA) ANSI X3.270-1996

SCSI Direct Attach (FC-PLDA) NCITS TR-19:1998

Fabric Loop Attachment (FC-FLA), NCITS TR-20:1998

Notes In N_Port point-to-point topologies, FCP error recovery does not work across events such as link bounce/cable pull. I/O to devices with FCP-2/FCP-3 support (for example, FC tape drives) will be disrupted by such events.

Name FSS – Fair share scheduler

Description The fair share scheduler (FSS) guarantees application performance by explicitly allocating shares of CPU resources to projects. A share indicates a project's entitlement to available CPU resources. Because shares are meaningful only in comparison with other project's shares, the absolute quantity of shares is not important. Any number that is in proportion with the desired CPU entitlement can be used.

The goals of the FSS scheduler differ from the traditional time-sharing scheduling class (TS). In addition to scheduling individual LWPs, the FSS scheduler schedules projects against each other, making it impossible for any project to acquire more CPU cycles simply by running more processes concurrently.

A project's entitlement is individually calculated by FSS independently for each processor set if the project contains processes bound to them. If a project is running on more than one processor set, it can have different entitlements on every set. A project's entitlement is defined as a ratio between the number of shares given to a project and the sum of shares of all active projects running on the same processor set. An active project is one that has at least one running or runnable process. Entitlements are recomputed whenever any project becomes active or inactive, or whenever the number of shares is changed.

Processor sets represent virtual machines in the FSS scheduling class and processes are scheduled independently in each processor set. That is, processes compete with each other only if they are running on the same processor set. When a processor set is destroyed, all processes that were bound to it are moved to the default processor set, which always exists. Empty processor sets (that is, sets without processors in them) have no impact on the FSS scheduler behavior.

If a processor set contains a mix of TS/IA and FSS processes, the fairness of the FSS scheduling class can be compromised because these classes use the same range of priorities. Fairness is most significantly affected if processes running in the TS scheduling class are CPU-intensive and are bound to processors within the processor set. As a result, you should avoid having processes from TS/IA and FSS classes share the same processor set. RT and FSS processes use disjoint priority ranges and therefore can share processor sets.

As projects execute, their CPU usage is accumulated over time. The FSS scheduler periodically decays CPU usages of every project by multiplying it with a decay factor, ensuring that more recent CPU usage has greater weight when taken into account for scheduling. The FSS scheduler continually adjusts priorities of all processes to make each project's relative CPU usage converge with its entitlement.

While FSS is designed to fairly allocate cycles over a long-term time period, it is possible that projects will not receive their allocated shares worth of CPU cycles due to uneven demand. This makes one-shot, instantaneous analysis of FSS performance data unreliable.

Note that share is not the same as utilization. A project may be allocated 50% of the system, although on the average, it uses just 20%. Shares serve to cap a project's CPU usage only when

there is competition from other projects running on the same processor set. When there is no competition, utilization may be larger than entitlement based on shares. Allocating a small share to a busy project slows it down but does not prevent it from completing its work if the system is not saturated.

The configuration of CPU shares is managed by the name server as a property of the `project(4)` database. In the following example, an entry in the `/etc/project` file sets the number of shares for project `x-files` to 10:

```
x-files:100::::project.cpu-shares=(privileged,10,none)
```

Projects with undefined number of shares are given one share each. This means that such projects are treated with equal importance. Projects with 0 shares only run when there are no projects with non-zero shares competing for the same processor set. The maximum number of shares that can be assigned to one project is 65535.

You can use the `prctl(1)` command to determine the current share assignment for a given project:

```
$ prctl -n project.cpu-shares -i project x-files
```

or to change the amount of shares if you have root privileges:

```
# prctl -r -n project.cpu-shares -v 5 -i project x-files
```

See the `prctl(1)` man page for additional information on how to modify and examine resource controls associated with active processes, tasks, or projects on the system. See `resource_controls(5)` for a description of the resource controls supported in the current release of the Solaris operating system.

By default, project `system` (project ID 0) includes all system daemons started by initialization scripts and has an "unlimited" amount of shares. That is, it is always scheduled first no matter how many shares are given to other projects.

The following command sets FSS as the default scheduler for the system:

```
# dispadmin -d FSS
```

This change will take effect on the next reboot. Alternatively, you can move processes from the time-share scheduling class (as well as the special case of init) into the FSS class without changing your default scheduling class and rebooting by becoming `root`, and then using the `priocntl(1)` command, as shown in the following example:

```
# priocntl -s -c FSS -i class TS
# priocntl -s -c FSS -i pid 1
```

Configuring Scheduler With Dispadmin

You can use the `dispadmin(1M)` command to examine and tune the FSS scheduler's time quantum value. Time quantum is the amount of time that a thread is allowed to run before it must relinquish the processor. The following example dumps the current time quantum for the fair share scheduler:

```
$ dispadmin -g -c FSS
    #
    # Fair Share Scheduler Configuration
    #
    RES=1000
    #
    # Time Quantum
    #
    QUANTUM=110
```

The value of the QUANTUM represents some fraction of a second with the fractional value determied by the reciprocal value of RES. With the default value of RES = 1000, the reciprocal of 1000 is .001, or milliseconds. Thus, by default, the QUANTUM value represents the time quantum in milliseconds.

If you change the RES value using `dispadmin` with the `-r` option, you also change the QUANTUM value. For example, instead of quantum of 110 with RES of 1000, a quantum of 11 with a RES of 100 results. The fractional unit is different while the amount of time is the same.

You can use the `-s` option to change the time quantum value. Note that such changes are not preserved across reboot. Please refer to the `dispadmin(1M)` man page for additional information.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SUNWcsu

See Also `prctl(1)`, `priocntl(1)`, `dispadmin(1M)`, `psrset(1M)`, `priocntl(2)`, `project(4)`, `attributes(5)`, `resource_controls(5)`

System Administration Guide: Virtualization Using the Solaris Operating System

Name ge – GEM Gigabit-Ethernet device driver

Synopsis `/dev/ge`

Description The `ge` Gigabit-Ethernet driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface, `dlpi(7P)` over GEM, SBus and PCI Gigabit-Ethernet add-in adapters. Multiple GEM-based adapters installed within the system are supported by the driver. The `ge` driver provides basic support for the GEM-based Ethernet hardware and handles the `SUNW,sbus-gem` (SBus GEM) and `pci108e,2bad` (PCI GEM) devices. Functions include chip initialization, frame transmit and receive, multicast and promiscuous support, and error recovery and reporting.

The `GEM` device provides 1000BASE-SX networking interfaces using the GEM ASIC external SERDES and fiber optical transceiver. The GEM ASIC provides the appropriate bus interface, MAC functions and physical code sub-layer (PCS) functions. The external SERDES connects to a fiber transceiver and provides the physical connection.

The 1000Base-SX standard specifies an auto-negotiation protocol to automatically select the mode of operation. In addition to duplex operation, the GEM ASIC can auto-negotiate for *IEEE 802.3x* frame based flow control capabilities. The GEM PCS is capable of performing auto-negotiation using the remote (or link partner) link end and receives the capabilities of the remote end. It selects the highest common demoninator mode of operation based on priorities. The `ge` driver also supports forced-mode operation under which the driver selects the mode of operation.

Application Programming Interface

ge and DLPI The cloning character-special device `/dev/ge` is used to access all `ge` controllers installed within the system.

The `ge` driver is a Style 2 data link service provider. All `M_PROTO` and `M_PCPROTO` type messages are interpreted as DLPI primitives. Valid DLPI primitives are defined in `<sys/dlpi.h>`. Refer to `dlpi(7P)` for more information.

You must send an explicit `DL_ATTACH_REQ` message to associate the opened stream with a particular device (`ppa`). The `ppa` ID is interpreted as an `unsigned long` data type and indicates the corresponding device instance (unit) number. An error (`DL_ERROR_ACK`) is returned by the driver if the `ppa` field value does not correspond to a valid device instance number for the system. The device is initialized on first attach and de-initialized (stopped) upon last detach.

The values returned by the driver in the `DL_INFO_ACK` primitive in response to the `DL_INFO_REQ` are:

- Maximum SDU is `1500` (`ETHERMTU` - defined in `<sys/ethernet.h>`).
- Minimum SDU is `0`.
- `dlsap` address length is `8`.
- MAC type is `DL_ETHER`.
- `sap` length value is –2, meaning the physical address component is followed immediately by a 2 byte `sap` component within the DLSAP address.

- Service mode is DL_CLDLS.
- Quality of service (QOS) is not supported; accordingly, QOS fields are 0.
- Provider style is DL_STYLE2.
- Version is DL_VERSION_2.
- Broadcast address value is Ethernet/IEEE broadcast address (0xFFFFFF).

Once in the DL_ATTACHED state, you must send a DL_BIND_REQ to associate a particular Service Access Pointer (SAP) with the stream. The ge driver interprets the sap field within the DL_BIND_REQ as an Ethernet *type*; accordingly, valid values for the sap field are in the [0-0xFFFF] range. Only one Ethernet type can be bound to the stream at any time.

If you select a sap with a value of 0, the receiver will be in *802.3* mode. All frames received from the media with a type field in the range [0-1500] are assumed to be *802.3* frames and are routed up all open streams bound to sap value 0. If more than one stream is in *802.3* mode, the frame will be duplicated and routed up multiple streams as DL_UNITDATA_IND messages.

In transmission, the driver checks the sap field of the DL_BIND_REQ to determine if the sap value is 0 and the destination type field is in the range [0-1500]. If either is true, the driver computes the length of the message, not including initial M_PROTO mblk (message block), of all subsequent DL_UNITDATA_REQ messages and transmits *802.3* frames of that value in the MAC frame header length field.

The ge driver DLSAP address format consists of the 6 byte physical (Ethernet) address component followed immediately by the 2 byte sap (type) component producing an 8 byte DLSAP address. Applications should *not* hard code to this particular implementation-specific DLSAP address format, but use information returned in the DL_INFO_ACK primitive to compose and decompose DLSAP addresses. The sap length, full DLSAP length and sap physical ordering are included within the DL_INFO_ACK. The physical address length can be computed by subtracting the sap length from the full DLSAP address length or by issuing the DL_PHYS_ADDR_REQ to obtain the current physical address associated with the stream.

Once in the DL_BOUND state, you may transmit frames on the Ethernet by sending DL_UNITDATA_REQ messages to the ge driver. The ge driver will route received Ethernet frames up all open and bound streams having a sap which matches the Ethernet type as DL_UNITDATA_IND messages. Received Ethernet frames are duplicated and routed up multiple open streams if necessary. The DLSAP address contained within the DL_UNITDATA_REQ and DL_UNITDATA_IND messages consists of both the sap (type) and physical (Ethernet) components.

In addition to the mandatory connectionless DLPI message set, the driver additionally supports ge primitives.

ge Primitives

The DL_ENABMULTI_REQ and DL_DISABMULTI_REQ primitives enable/disable reception of individual multicast group addresses. A set of multicast addresses may be iteratively created and modified on a per-stream basis using these primitives. The DL_ENABMULTI_REQ and DL_DISABMULTI_REQ primitives are accepted by the driver in any state following DL_ATTACHED.

With the DL_PROMISC_PHYS flag set in the dl_level field, the DL_PROMISCON_REQ and DL_PROMISCOFF_REQ primitives enable/disable reception of all promiscuous mode frames on the media including frames generated by the local host. When used with the DL_PROMISC_SAP flag set, this enables/disables reception of all sap (Ethernet type) values. When used with the DL_PROMISC_MULTI flag set, this enables/disables reception of all multicast group addresses. The effect of each is always on a per-stream basis and independent of the other sap and physical level configurations on the stream or other streams.

The DL_PHYS_ADDR_REQ primitive returns the six octet Ethernet address currently associated (attached) to the stream in the DL_PHYS_ADDR_ACK primitive. This primitive is valid only in states following a successful DL_ATTACH_REQ.

The DL_SET_PHYS_ADDR_REQ primitive changes the 6 octet Ethernet address currently associated (attached) to the stream. The credentials of the process which originally opened the stream must be superuser or EPERM is returned in the DL_ERROR_ACK. The DL_SET_PHYS_ADDR_REQ primitive is destructive and affects all other current and future streams attached to this device. A M_ERROR is sent up all other streams attached to the device when DL_SET_PHYS_ADDR_REQ is successful on the stream. Once changed, all streams subsequently opened and attached to the device will obtain the new physical address. Once changed, the physical address will remain until DL_SET_PHYS_ADDR_REQ is used to change the physical address again or the system is rebooted, whichever comes first.

ge DRIVER

By default, the ge driver performs auto-negotiation to select the mode and flow control capabilities of the link. The link can be in one of the following modes:

- 1000 Mbps, full-duplex
- 1000 Mbps, half-duplex
- Symmetric pause
- Asymmetric pause

Speeds and modes are described in the 1000Base-TX standard.

The auto-negotation protocol automatically selects:

- Operation mode (half-duplex or full-duplex)
- Flow control capability (Symmetric and/or Asymmetric)

The auto–negotiation protocol:

- Gets all the modes of operation supported by the link partner.
- Advertises its capabilities to the link partner.

- Selects the highest common denominator mode of operation based on the priorities.

When by default, auto-negotiation is used to bring up the link and select the common mode of operation with the link partner, the GEM hardware is capable of all of the operating modes listed above. The PCS also supports forced-mode of operation under which the driver can select the mode of operation and flow control capabilities using the `ndd` utility.

The `GEM` device also supports programmable Inter-Packet Gap (IPG) parameters `ipg1` and `ipg2`. By default, the driver sets `ipg1` to 8 byte-times and `ipg2` to 4 byte-times, (the standard values.) You may want to alter these values from the standard 1000 Mpbs `IPG` set to 0.096 microseconds.

ge Parameter List

You can use the `ge` driver to set and get parameters for the GEM device. The parameter list includes current transceiver status, current link status, inter-packet gap, PCS capabilities and link partner capabilities.

The PCS has two set of capabilities. One set reflects the capabilities of the hardware and are read-only. The second set are read/write and are used in speed selection and reflect the values you choose. At boot time, both sets will be the same. The link partner capabilities are read only and cannot be modified.

Files

`/dev/ge`	`ge` special character device
`/kernel/drv/ge.conf`	System wide default device driver properties

See Also

`ndd`(1M), `netstat`(1M), `driver.conf`(4), `dlpi`(7P), `hme`(7D), `qfe`(7d)

Name gld – Generic LAN Driver

Synopsis

```
#include <sys/stropts.h>

#include <sys/stream.h>

#include <sys/dlpi.h>

#include <sys/gld.h>
```

Interface Level Solaris architecture specific (Solaris DDI).

Description GLD is a multi-threaded, clonable, loadable kernel module providing support for Solaris local area network (LAN) device drivers. LAN drivers in Solaris are STREAMS-based drivers that use the Data Link Provider Interface (DLPI) to communicate with network protocol stacks. These protocol stacks use the network drivers to send and receive packets on a local area network. A network device driver must implement and adhere to the requirements imposed by the DDI/DKI specification, STREAMS specification, DLPI specification, and programmatic interface of the device itself.

GLD implements most STREAMS and DLPI functionality required of a Solaris LAN driver. Several Solaris network drivers are implemented using GLD.

A Solaris network driver implemented using GLD comprises two distinct parts: a generic component that deals with STREAMS and DLPI interfaces, and a device-specific component that deals with the particular hardware device. The device-specific module indicates its dependency on the GLD module and registers itself with GLD from within the driver's `attach(9E)` function. Once it is successfully loaded, the driver is DLPI-compliant. The device-specific part of the driver calls `gld(9F)` functions when it receives data or needs some service from GLD. GLD makes calls into the `gld(9E)` entry points of the device-specific driver through pointers provided to GLD by the device-specific driver when it registered itself with GLD. The `gld_mac_info(9S)` structure is the main data interface between GLD and the device-specific driver.

The GLD facility currently supports devices of type `DL_ETHER`, `DL_TPR`, and `DL_FDDI`. GLD drivers are expected to process fully-formed MAC-layer packets and should not perform logical link control (LLC) handling.

Note – Support for the DL_TPR and DL_FDDI media types in GLD is obsolete and may be removed in a future release of Solaris.

In some cases, it may be necessary or desirable to implement a full DLPI-compliant driver without using the GLD facility. This is true for devices that are not IEEE 802-style LAN devices, or where a device type or DLPI service not supported by GLD is required.

Device Naming Constraints The name of the device-specific driver module must adhere to the naming constraints outlined in the NOTES section of `dlpi(7P)`.

Type DL_ETHER: Ethernet V2 and ISO 8802-3 (IEEE 802.3)

For devices designated type DL_ETHER, GLD provides support for both Ethernet V2 and ISO 8802-3 (IEEE 802.3) packet processing. Ethernet V2 enables a data link service user to access and use any of a variety of conforming data link service providers without special knowledge of the provider's protocol. A service access point (SAP) is the point through which the user communicates with the service provider.

SAP 0 denotes that the user wishes to use *802.3* mode. In transmission, GLD checks the destination SAP value of the DL_UNITDATA_REQ and the SAP value to which the stream is bound. If both are 0, the GLD computes the length of the packet payload and transmits *802.3* frames having that length in the MAC frame header type field. Such lengths will never exceed 1500.

All frames received from the media that have a type field in the range [0-1500] are assumed to be *802.3* frames and are routed up all open streams that are in *802.3* mode, (those streams bound to a SAP value in of 0. If more than one stream is in *802.3* mode, the incoming frame is duplicated and routed up each such stream.

Streams bound to a SAP value of 1536 or greater receive incoming packets whose Ethernet MAC header type value exactly matches the value of the SAP to which the stream is bound. SAP values in the range [1-1535] are undefined and should not be used.

Types DL_TPR and DL_FDDI: SNAP Processing

Note – Support for the DL_TPR and DL_FDDI media types in GLD is obsolete and may be removed in a future release of Solaris.

For media types DL_TPR and DL_FDDI, GLD implements minimal SNAP (Sub-Net Access Protocol) processing for SAP values of 1536 or greater. A SAP value of 0 denotes that the user wishes to use LLC mode. SAP values in the range [1-1535] have undefined semantics and should not be used.

SNAP headers are carried under LLC headers with destination SAP 0xAA. For outgoing packets with SAP values greater than 1535, GLD creates an LLC+SNAP header that always looks like:

"AA AA 03 00 00 00 XX XX"

where "XX XX" represents the 16-bit SAP, corresponding to the Ethernet V2 style "type." This is the only class of SNAP header that is processed - non-zero OUI fields, and LLC control fields other than 03 are considered to be LLC packets with SAP 0xAA.

A DL_UNITDATA_REQ message specifying a destination SAP value of 0, passed down a stream bound to SAP 0, is assumed to contain an LLC packet and will not undergo SNAP processing.

Incoming packets are examined to ascertain whether they fall into the format specified above. Packets that do will be passed to streams bound to the packet's 16-bit SNAP type, as well as being passed to any stream in LLC mode (those bound to a SAP value of 0).

Type DL_TPR: Source Routing

Note – Support for the DL_TPR media type in GLD is obsolete and may be removed in a future release of Solaris.

For type `DL_TPR` devices, GLD implements minimal support for source routing. Source routing enables a station that is sending a packet across a bridged medium to specify (in the packet MAC header) routing information that determines the route that the packet will take through the network.

Functionally, the source routing support provided by GLD learns routes, solicits and responds to requests for information about possible multiple routes and selects among the multiple routes that are available. It adds *Routing Information Fields* to the MAC headers of outgoing packets and recognizes such fields in incoming packets.

GLD's source routing support does not implement the full *Route Determination Entity* (RDE) specified in *ISO 8802-2 (IEEE 802.2)* Section 9. However, it is designed to interoperate with any such implementations that may exist in the same (or a bridged) network.

Style 1 and 2 Providers

GLD implements both Style 1 and Style 2 providers. A physical point of attachment (PPA) is the point at which a system attaches itself to a physical communication medium. All communication on that physical medium funnels through the PPA. The Style 1 provider attaches the stream to a particular PPA based on the major/minor device that has been opened. The Style 2 provider requires the DLS user to explicitly identify the desired PPA using `DL_ATTACH_REQ`. In this case, `open(9E)` creates a stream between the user and GLD and `DL_ATTACH_REQ` subsequently associates a particular PPA with that stream. Style 2 is denoted by a minor number of zero. If a device node whose minor number is not zero is opened, Style 1 is indicated and the associated PPA is the minor number minus 1. In both Style 1 and Style 2 opens, the device is cloned.

Implemented DLPI Primitives

GLD implements the following DLPI primitives:

The `DL_INFO_REQ` primitive requests information about the DLPI stream. The message consists of one `M_PROTO` message block. GLD returns device-dependent values in the `DL_INFO_ACK` response to this request, based on information the GLD-based driver specified in the `gld_mac_info(9S)` structure passed to `gld_register()`. However GLD returns the following values on behalf of all GLD-based drivers:

- The version is `DL_VERSION_2`.
- The service mode is `DL_CLDLS` — GLD implements connectionless-mode service.
- The provider style is `DL_STYLE1` or `DL_STYLE2`, depending on how the stream was opened.

The `DL_ATTACH_REQ` primitive is called to associate a PPA with a stream. This request is needed for Style 2 DLS providers to identify the physical medium over which the communication will transpire. Upon completion, the state changes from `DL_UNATTACHED` to `DL_UNBOUND`. The

message consists of one M_PROTO message block. This request may not be issued when using the driver in Style 1 mode; streams opened using Style 1 are already attached to a PPA by the time the open completes.

The DL_DETACH_REQ primitive requests to detach the PPA from the stream. This is only allowed if the stream was opened using Style 2.

The DL_BIND_REQ and DL_UNBIND_REQ primitives bind and unbind a DLSAP to the stream. The PPA associated with each stream will have been initialized upon completion of the processing of the DL_BIND_REQ. Multiple streams may be bound to the same SAP; each such stream receives a copy of any packets received for that SAP.

The DL_ENABMULTI_REQ and DL_DISABMULTI_REQ primitives enable and disable reception of individual multicast group addresses. A set of multicast addresses may be iteratively created and modified on a per-stream basis using these primitives. The stream must be attached to a PPA for these primitives to be accepted.

The DL_PROMISCON_REQ and DL_PROMISCOFF_REQ primitives enable and disable promiscuous mode on a per-stream basis, either at a physical level or at the SAP level. The DL Provider will route all received messages on the media to the DLS user until either a DL_DETACH_REQ or a DL_PROMISCOFF_REQ is received or the stream is closed. Physical level promiscuous mode may be specified for all packets on the medium or for multicast packets only. The stream must be attached to a PPA for these primitives to be accepted.

The DL_UNITDATA_REQ primitive is used to send data in a connectionless transfer. Because this is an unacknowledged service, there is no guarantee of delivery. The message consists of one M_PROTO message block followed by one or more M_DATA blocks containing at least one byte of data.

The DL_UNITDATA_IND type is used when a packet is received and is to be passed upstream. The packet is put into an M_PROTO message with the primitive set to DL_UNITDATA_IND.

The DL_PHYS_ADDR_REQ primitive returns the MAC address currently associated with the PPA attached to the stream, in the DL_PHYS_ADDR_ACK primitive. When using style 2, this primitive is only valid following a successful DL_ATTACH_REQ.

The DL_SET_PHYS_ADDR_REQ primitive changes the MAC address currently associated with the PPA attached to the stream. This primitive affects all other current and future streams attached to this device. Once changed, all streams currently or subsequently opened and attached to this device will obtain this new physical address. The new physical address will remain in effect until this primitive is used to change the physical address again or the driver is reloaded.

The DL_GET_STATISTICS_REQ primitive requests a DL_GET_STATISTICS_ACK response containing statistics information associated with the PPA attached to the stream. Style 2 streams must be attached to a particular PPA using DL_ATTACH_REQ before this primitive will be successful.

GLD supports the DL_NOTE_LINK_UP, DL_NOTE_LINK_DOWN and DL_NOTE_SPEED notifications using the DL_NOTIFY_IND primitive. See dlpi(7P).

Implemented ioctl Functions

GLD implements the DLIOCRAW ioctl described in dlpi(7P). For any other ioctl command, GLD passes it to the device-specific driver's gldm_ioctl() function as described in gld(9E).

Requirements on GLD Drivers

GLD-based drivers must include the header file <sys/gld.h>.

GLD-based drivers must also specify a link dependency on "misc/gld". (See the -N option in ld(1)).

GLD implements the open(9E) and close(9E) functions and the required STREAMS put(9E) and srv(9E) functions on behalf of the device-specific driver. GLD also implements the getinfo(9E) function for the driver.

The mi_idname element of the module_info(9S) structure is a string specifying the name of the driver. This must exactly match the name of the driver module as it exists in the file system.

The read-side qinit(9S) structure should specify the following elements as shown below:

```
qi_putp      NULL

qi_srvp      gld_rsrv

qi_qopen     gld_open

qi_qclose    gld_close
```

The write-side qinit(9S) structure should specify the following elements as shown below:

```
qi_putp      gld_wput

qi_srvp      gld_wsrv

qi_qopen     NULL

qi_qclose    NULL
```

The devo_getinfo element of the dev_ops(9S) structure should specify gld_getinfo as the getinfo(9E) routine.

The driver's attach(9E) function does all the work of associating the hardware-specific device driver with the GLD facility and preparing the device and driver for use.

The attach(9E) function allocates a gld_mac_info(9S) ("macinfo") structure using gld_mac_alloc(). The driver usually needs to save more information per device than is defined in the macinfo structure; it should allocate the additional required data structure and save a pointer to it in the gldm_private member of the gld_mac_info(9S) structure.

The attach(9E) routine must initialize the macinfo structure as described in gld_mac_info(9S) and then call gld_register() to link the driver with the GLD module. The driver should map registers if necessary and be fully initialized and prepared to accept interrupts before calling gld_register(). The attach(9E) function should add interrupts but not enable the device to generate them. The driver should reset the hardware before calling gld_register() to ensure it is quiescent; the device must not be started or put into a state where it may generate an interrupt before gld_register() is called. That will be done later when GLD calls the driver's gldm_start() entry point described in gld(9E). Once gld_register() succeeds, the gld(9E) entry points may be called by GLD at any time.

The attach(9E) routine should return DDI_SUCCESS if gld_register() succeeds. If gld_register() fails, it returns DDI_FAILURE and the attach(9E) routine should deallocate any resources it allocated before calling gld_register() and then also return DDI_FAILURE. Under no circumstances should a failed macinfo structure be reused; it should be deallocated using gld_mac_free().

The detach(9E) function should attempt to unregister the driver from GLD. This is done by calling gld_unregister() described in gld(9F). The detach(9E) routine can get a pointer to the needed gld_mac_info(9S) structure from the device's private data using ddi_get_driver_private(9F). gld_unregister() checks certain conditions that could require that the driver not be detached. If the checks fail, gld_unregister() returns DDI_FAILURE, in which case the driver's detach(9E) routine must leave the device operational and return DDI_FAILURE. If the checks succeed, gld_unregister() ensures that the device interrupts are stopped, calling the driver's gldm_stop() routine if necessary, unlinks the driver from the GLD framework, and returns DDI_SUCCESS. In this case, the detach(9E) routine should remove interrupts, deallocate any data structures allocated in the attach(9E) routine, using gld_mac_free() to deallocate the macinfo structure, and return DDI_SUCCESS. It is important to remove the interrupt *before* calling gld_mac_free().

Network Statistics

Solaris network drivers must implement statistics variables. GLD itself tallies some network statistics, but other statistics must be counted by each GLD-based driver. GLD provides support for GLD-based drivers to report a standard set of network driver statistics. Statistics are reported by GLD using the kstat(7D) and kstat(9S) mechanism. The DL_GET_STATISTICS_REQ DLPI command may also be used to retrieve the current statistics counters. All statistics are maintained as unsigned, and all are 32 bits unless otherwise noted.

GLD maintains and reports the following statistics.

rbytes64	Total bytes successfully received on the interface (64 bits).
rbytes	Total bytes successfully received on the interface.

`obytes64`	Total bytes requested to be transmitted on the interface (64 bits).
`obytes`	Total bytes requested to be transmitted on the interface.
`ipackets64`	Total packets successfully received on the interface (64 bits).
`ipackets`	Total packets successfully received on the interface.
`opackets64`	Total packets requested to be transmitted on the interface (64 bits).
`opackets`	Total packets requested to be transmitted on the interface.
`multircv`	Multicast packets successfully received, including group and functional addresses (long).
`multixmt`	Multicast packets requested to be transmitted, including group and functional addresses (long).
`brdcstrcv`	Broadcast packets successfully received (long).
`brdcstxmt`	Broadcast packets requested to be transmitted (long).
`unknowns`	Valid received packets not accepted by any stream (long).
`noxmtbuf`	Packets discarded on output because transmit buffer was busy, or no buffer could be allocated for transmit (long).
`blocked`	Times a received packet could not be put up a stream because the queue was flow controlled (long).
`xmtretry`	Times transmit was retried after having been delayed due to lack of resources (long).
`promisc`	Current “promiscuous" state of the interface (string).

The device dependent driver counts the following statistics, keeping track of them in a private per-instance structure. When GLD is asked to report statistics, it calls the driver's `gldm_get_stats()` entry point, as described in `gld(9E)`, to update the device-specific statistics in the `gld_stats(9S)` structure. GLD then reports the updated statistics using the named statistics variables below.

`ifspeed`	Current estimated bandwidth of the interface in bits per second (64 bits).
`media`	Current media type in use by the device (string).
`intr`	Times interrupt handler was called and claimed the interrupt (long).
`norcvbuf`	Times a valid incoming packet was known to have been discarded because no buffer could be allocated for receive (long).
`ierrors`	Total packets received that couldn't be processed because they contained errors (long).
`oerrors`	Total packets that weren't successfully transmitted because of errors (long).

`missed`	Packets known to have been dropped by the hardware on receive (long).
`uflo`	Times FIFO underflowed on transmit (long).
`oflo`	Times receiver overflowed during receive (long).

The following group of statistics applies to networks of type `DL_ETHER`; these are maintained by device-specific drivers of that type, as above.

`align_errors`	Packets received with framing errors (not an integral number of octets) (long).
`fcs_errors`	Packets received with CRC errors (long).
`duplex`	Current duplex mode of the interface (string).
`carrier_errors`	Times carrier was lost or never detected on a transmission attempt (long).
`collisions`	Ethernet collisions during transmit (long).
`ex_collisions`	Frames where excess collisions occurred on transmit, causing transmit failure (long).
`tx_late_collisions`	Times a transmit collision occurred late (after 512 bit times) (long).
`defer_xmts`	Packets without collisions where first transmit attempt was delayed because the medium was busy (long).
`first_collisions`	Packets successfully transmitted with exactly one collision.
`multi_collisions`	Packets successfully transmitted with multiple collisions.
`sqe_errors`	Times SQE test error was reported.
`macxmt_errors`	Packets encountering transmit MAC failures, except carrier and collision failures.
`macrcv_errors`	Packets received with MAC errors, except align, fcs, and toolong errors.
`toolong_errors`	Packets received larger than the maximum permitted length.
`runt_errors`	Packets received smaller than the minimum permitted length (long).

The following group of statistics applies to networks of type `DL_TPR`; these are maintained by device-specific drivers of that type, as above.

`line_errors`	Packets received with non-data bits or FCS errors.
`burst_errors`	Times an absence of transitions for five half-bit timers was detected.

`signal_losses`	Times loss of signal condition on the ring was detected.
`ace_errors`	Times an AMP or SMP frame in which A is equal to C is equal to 0, was followed by another such SMP frame without an intervening AMP frame.
`internal_errors`	Times the station recognized an internal error.
`lost_frame_errors`	Times the TRR timer expired during transmit.
`frame_copied_errors`	Times a frame addressed to this station was received with the FS field A bit set to 1.
`token_errors`	Times the station acting as the active monitor recognized an error condition that needed a token transmitted.
`freq_errors`	Times the frequency of the incoming signal differed from the expected frequency.

The following group of statistics applies to networks of type `DL_FDDI`; these are maintained by device-specific drivers of that type, as above.

`mac_errors`	Frames detected in error by this MAC that had not been detected in error by another MAC.
`mac_lost_errors`	Frames received with format errors such that the frame was stripped.
`mac_tokens`	Number of tokens received (total of non-restricted and restricted).
`mac_tvx_expired`	Number of times that TVX has expired.
`mac_late`	Number of TRT expirations since this MAC was reset or a token was received.
`mac_ring_ops`	Number of times the ring has entered the "Ring_Operational" state from the "Ring Not Operational" state.

Files `/kernel/misc/gld` loadable kernel module

See Also `ld(1)`, `kstat(7D)`, `dlpi(7P)`, `attach(9E)`, `gld(9E)`, `open(9E)`, `gld(9F)`, `gld_mac_info(9S)`, `gld_stats(9S)`, `kstat(9S)`

Writing Device Drivers

Warnings Contrary to the DLPI specification, GLD returns the device's correct address length and broadcast address in `DL_INFO_ACK` even before the stream has been attached to a PPA.

Promiscuous mode may only be entered by streams that are attached to a PPA.

The physical address of a PPA may be changed by the superuser while other streams are bound to the same PPA.

Name glm – GLM SCSI Host Bus Adapter Driver

Synopsis `scsi@unit-address`

Description The `glm` Host Bus Adapter driver is a SCSA compliant nexus driver that supports the LSI 53c810, LSI 53c875, LSI 53c876, LSI 53C896 and LSI 53C1010 SCSI chips

It supports the standard functions provided by the SCSA interface. That is, it supports tagged and untagged queuing, Narrow/Wide/Fast/Ultra SCSI/Ultra SCSI 2/Ultra SCSI 3, and auto request sense, but it does not support linked commands.

Driver Configuration Configure the `glm` driver by defining properties in `glm.conf`. These properties override the global SCSI settings. `glm` supports these properties which can be modified by the user: `scsi-options`, `target`<*n*>`-scsi-options`, `scsi-reset-delay`, `scsi-tag-age-limit`, `scsi-watchdog-tick`, and `scsi-initiator-id`.

`target`<*n*>`-scsi-options` overrides the `scsi-options` property value for `target`<*n*>. <*n*> can vary from decimal `0` to `15`. `glm` supports these `scsi-options`: `SCSI_OPTIONS_DR`, `SCSI_OPTIONS_SYNC`, `SCSI_OPTIONS_TAG`, `SCSI_OPTIONS_FAST`, `SCSI_OPTIONS_WIDE`, `SCSI_OPTIONS_FAST20`, `SCSI_OPTIONS_FAST40` and `SCSI_OPTIONS_FAST80`.

After periodic interval `scsi-watchdog-tick`, the `glm` driver searches through all current and disconnected commands for timeouts.

`scsi-tag-age-limit` is the number of times that the `glm` driver attempts to allocate a particular tag ID that is currently in use after going through all tag IDs in a circular fashion. After finding the same tag ID in use `scsi-tag-age-limit` times, no more commands will be submitted to this target until all outstanding commands complete or timeout.

Refer to `scsi_hba_attach(9F)`.

Examples **EXAMPLE 1** Using the `glm` Configuration File

Create a file called `/kernel/drv/glm.conf` and add the following line:

```
scsi-options=0x78;
```

This disables tagged queuing, Fast/Ultra SCSI and wide mode for all `glm` instances.

The following example disables an option for one specific `glm` (refer to `driver.conf(4)` and `pci(4)` for more details):

```
name="glm" parent="/pci@1f,4000"
    unit-address="3"
    target1-scsi-options=0x58
    scsi-options=0x178 scsi-initiator-id=6;
```

EXAMPLE 1 Using the `glm` Configuration File *(Continued)*

Note that the default initiator ID in OBP is 7 and that the change to ID 6 will occur at attach time. It may be preferable to change the initiator ID in OBP.

The example above sets `scsi-options` for target 1 to `0x58` and all other targets on this SCSI bus to `0x178`.

The physical pathname of the parent can be determined using the `/devices` tree or following the link of the logical device name:

```
# ls -l /dev/rdsk/c0t0d0s0
lrwxrwxrwx  1 root   root      45 May 16 10:08 /dev/rdsk/c0t0d0s0 ->
   . . / . . /devices/pci@1f,4000/scsi@3/sd@0,0:a,raw
```

In this case, like the example above, the parent is `/pci@1f,4000` and the `unit-address` is the number bound to the `scsi@3` node.

To set `scsi-options` more specifically per target:

```
target1-scsi-options=0x78;
device-type-scsi-options-list =
    "SEAGATE ST32550W", "seagate-scsi-options" ;
seagate-scsi-options = 0x58;
scsi-options=0x3f8;
```

The above sets `scsi-options` for target 1 to `0x78` and for all other targets on this SCSI bus to `0x3f8` except for one specific disk type which will have `scsi-options` set to `0x58`.

`scsi-options` specified per target ID have the highest precedence, followed by `scsi-options` per device type. Global `scsi-options` (for all `glm` instances) per bus have the lowest precedence.

The system needs to be rebooted before the specified `scsi-options` take effect.

Driver Capabilities

The target driver needs to set capabilities in the `glm` driver in order to enable some driver features. The target driver can query and modify these capabilities: `synchronous`, `tagged-qing`, `wide-xfer`, `auto-rqsense`, `qfull-retries`, `qfull-retry-interval`. All other capabilities can only be queried.

By default, `tagged-qing`, `auto-rqsense`, and `wide-xfer` capabilities are disabled, while `disconnect`, `synchronous`, and `untagged-qing` are enabled. These capabilities can only have binary values (`0` or `1`). The default value for `qfull-retries` is `10` and the default value for `qfull-retry-interval` is `100`. The `qfull-retries` capability is a `uchar_t` (`0` to `255`) while `qfull-retry-interval` is a `ushort_t` (`0` to `65535`).

The target driver needs to enable `tagged-qing` and `wide-xfer` explicitly. The `untagged-qing` capability is always enabled and its value cannot be modified.

Whenever there is a conflict between the value of `scsi-options` and a capability, the value set in `scsi-options` prevails. Only `whom != 0` is supported in the `scsi_ifsetcap(9F)` call.

Refer to `scsi_ifsetcap(9F)` and `scsi_ifgetcap(9F)` for details.

Files

`/kernel/drv/glm`	32–bit ELF kernel module (x86).
`/kernel/drv/amd64/glm`	64–bit ELF kernel module (x86).
`/kernel/drv/sparcv9/glm`	64–bit ELF kernel module (SPARC).
`/kernel/drv/glm.conf`	Optional configuration file

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	Limited to PCI-based systems with LSI 53c810, LSI 53c875, LSI 53c876, LSI 53c896 and LSI 53c1010 SCSI I/O processors

See Also

`prtconf(1M)`, `driver.conf(4)`, `pci(4)`, `attributes(5)`, `scsi_abort(9F)`, `scsi_hba_attach(9F)`, `scsi_ifgetcap(9F)`, `scsi_ifsetcap(9F)`, `scsi_reset(9F)`, `scsi_sync_pkt(9F)`, `scsi_transport(9F)`, `scsi_device(9S)`, `scsi_extended_sense(9S)`, `scsi_inquiry(9S)`, `scsi_pkt(9S)`

Writing Device Drivers

ANSI Small Computer System Interface-2 (SCSI-2),

LSI Logi Inc (formerly Symbios Logic Inc.):

- SYM53c810 PCI-SCSI I/O processor with Narrow operation
- SYM53c875 PCI-SCSI I/O Processor With Fast-20
- SYM53c876 PCI-SCSI I/O processor Dual channel Fast-20
- SYM53c896 PCI-SCSI I/O processor Dual channel Fast-40
- SYM53c1010 PCI-SCSI I/O processor Dual Channel Fast-80

Diagnostics

The messages described below are some that may appear on the system console, as well as being logged.

Device is using a hilevel intr
: The device was configured with an interrupt level that cannot be used with this `glm` driver. Check the PCI device.

map setup failed
: Driver was unable to map device registers; check for bad hardware. Driver did not attach to device; SCSI devices will be inaccessible.

glm_script_alloc failed
The driver was unable to load the SCRIPTS for the SCSI processor, check for bad hardware. Driver did not attach to device; SCSI devices will be inaccessible.

cannot map configuration space.
The driver was unable to map in the configuration registers. Check for bad hardware. SCSI devices will be inaccessible.

attach failed
The driver was unable to attach; usually preceded by another warning that indicates why attach failed. These can be considered hardware failures.

SCSI bus DATA IN phase parity error
The driver detected parity errors on the SCSI bus.

SCSI bus MESSAGE IN phase parity error
The driver detected parity errors on the SCSI bus.

SCSI bus STATUS phase parity error
The driver detected parity errors on the SCSI bus.

Unexpected bus free
Target disconnected from the bus without notice. Check for bad hardware.

Disconnected command timeout for Target <id>.<lun>
A timeout occurred while target `id`/*lun* was disconnected. This is usually a target firmware problem. For tagged queuing targets, <*n*> commands were outstanding when the timeout was detected.

Disconnected tagged cmd(s) (<n>) timeout for Target <id>.<lun>
A timeout occurred while target `id`/*lun* was disconnected. This is usually a target firmware problem. For tagged queuing targets, <*n*> commands were outstanding when the timeout was detected.

Connected command timeout for Target <id>.<lun>
This is usually a SCSI bus problem. Check cables and termination.

Target <id> reducing sync. transfer rate
A data transfer hang or DATA-IN phase parity error was detected. The driver attempts to eliminate this problem by reducing the data transfer rate.

Target <id> reverting to async. mode
A second data transfer hang was detected for this target. The driver attempts to eliminate this problem by reducing the data transfer rate.

Target <id> disabled wide SCSI mode
A second data phase hang was detected for this target. The driver attempts to eliminate this problem by disabling wide SCSI mode.

auto request sense failed
An attempt to start an auto request packet failed. Another auto request packet may already be in transport.

invalid reselection (<id>.<lun>)
A reselection failed; target accepted `abort` or `reset`, but still tries to reconnect. Check for bad hardware.

invalid intcode
The SCRIPTS processor generated an invalid SCRIPTS interrupt. Check for bad hardware.

Notes The x4422a card uses an OBP (forth) firmware and is incompatible with x86 BIOS. As a result, the x4422a cannot be used as a boot device on x86.

The glm driver supports the following LSI chips:

- LSI 53C810, which supports Narrow, Fast SCSI mode. The maximum SCSI bandwidth is 10 MB/sec.
- LSI 53C875, which supports Wide, Fast, and Ultra SCSI mode. The maximum SCSI bandwidth is 40 MB/sec.
- LSI 53C896, which supports Wide, Fast and Ultra SCSI 2 mode. The maximum LVD SCSI bandwidth is 80 MB/sec.
- LSI 53c1010, which supports wide, Fast and Ultra SCSI 3 mode. The maximum LVD SCSI bandwidth is 160 MB/sec.

The `glm` driver exports properties indicating per target the negotiated transfer speed (`target<`*n*`>-sync-speed`), whether wide bus is supported (`target<`*n*`>-wide`), for that particular target (`target<`*n*`>-scsi-options`), and whether tagged queuing has been enabled (`target<`*n*`>-TQ`). The `sync-speed` property value is the data transfer rate in KB/sec. The `target<`*n*`>-TQ` and the `target<`*n*`>-wide` property have value `1` to indicate that the corresponding capability is enabled, or `0` to indicate that the capability is disabled for that target. Refer to `prtconf(1M)` (verbose option) for viewing the `glm` properties.

```
scsi, instance #0
    Driver properties:
        name <target6-TQ> length <4>
            value <0x00000000>.
        name <target6-wide> length <4>
            value <0x00000000>.
        name <target6-sync-speed> length <4>
            value <0x00002710>.
        name <target1-TQ> length <4>
            value <0x00000001>.
        name <target1-wide> length <4>
            value <0x00000000>.
        name <target1-sync-speed> length <4>
            value <0x00002710>.
```

```
name <target0-TQ> length <4>
    value <0x00000001>.
name <target0-wide> length <4>
    value <0x00000001>.
name <target0-sync-speed> length <4>
    value <0x00009c40>.
name <scsi-options> length <4>
    value <0x000007f8>.
name <scsi-watchdog-tick> length <4>
    value <0x0000000a>.
name <scsi-tag-age-limit> length <4>
    value <0x00000002>.
name <scsi-reset-delay> length <4>
    value <0x00000bb8>.
name <latency-timer> length <4>
    value <0x00000088>.
name <cache-line-size> length <4>
    value <0x00000010>.
```

Name gpio_87317 – General purpose I/O driver for SuperIO

Description The `gpio_87317` driver is the general purpose I/O driver for the National Semiconductor SuperIO (PC87317) chipset. It supports remote system controller (RSC) administration via an interface to the SuperIO's general purpose I/O bits.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	Limited to SPARC systems with SuperIO
Availability	SUNWcar
Interface Stability	Unstable

See Also *PC87317VUL/PC97317VUL SuperI/O Data Sheet* — National Semiconductor

Name grbeep – Platform-dependent beep driver for SMBus-based hardware

Synopsis `beep@unit-address`

Description The `grbeep` driver generates beeps on platforms (including Sun Blade 100, 150, 1500, 2500) that use SMBbus-based registers and USB keyboards. When the `KIOCCMD` ioctl is issued to the USB keyboard module (see `usbkbm(7M)`) with command `KBD_CMD_BELL/KBD_CMD_NOBELL`, `usbkbm(7M)` passes the request to the `grbeep` driver to turn the beep on and off, respectively.

Files `/platform/sun4u/kernel/drv/sparcv9/grbeep` 64–bit ELF kernel driver

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SMBus-based SPARC
Availability	SUNWcarx.u

See Also `kbd(1)`, `attributes(5)`, `bbc_beep(7D)`, `kb(7M)`, `usbkbm(7M)`

Writing Device Drivers

Diagnostics None

Name hci1394 – 1394 OpenHCI host controller driver

Synopsis

```
firewire@unit-address
```

Description The `hci1394` host controller driver is an IEEE 1394 compliant nexus driver that supports the *1394 Open Host Controller Interface Specification 1.0*, an industry standard developed by Sun, Apple, Compaq, Intel, Microsoft, National Semconductor, and Texas Instruments. The `hci1394` driver supports asynchronous transfers, isochronous transfers, and bus reset management. The `hci1394` driver also supports the nexus device control interface.

Files

`/kernel/drv/sparcv9/hci1394`	64–bit SPARC ELF kernel module
`/kernel/drv/hci1394`	32-bit x86 ELF kernel module
`/kernel/drv/amd64/hci1394`	64-bit x86 ELF kernel module

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86, PCI-based systems
Availability	SUNW1394x
Interface Stability	Unstable

See Also `attributes(5)`, `ieee1394(7D)`

IEEE 1394 - IEEE Standard for a High Performance Serial Bus

1394 Open Host Controller Interface Specification 1.0

Name hdio – SMD and IPI disk control operations

Synopsis

```
#include <sys/hdio.h>
```

Description **Note –** The SMC and IPI drivers have been discontinued. dkio(7I) is now the preferred method for retrieving disk information.

The SMD and IPI disk drivers supplied with this release support a set of ioctl(2) requests for diagnostics and bad sector information. Basic to these `ioctl()` requests are the definitions in `<sys/hdio.h>`.

ioctls

`HDKIOCGTYPE` The argument is a pointer to a `hdk_type` structure (described below). This `ioctl()` gets specific information from the hard disk.

`HDKIOCSTYPE` The argument is a pointer to a `hdk_type` structure (described below). This `ioctl()` sets specific information about the hard disk.

```
/*
 * Used for drive info
 */
struct hdk_type {
 ushort_t  hdkt_hsect;    /* hard sector count (read only) */
 ushort_t  hdkt_promrev;  /* prom revision (read only) */
 uchar_t   hdkt_drtype;   /* drive type (ctlr specific) */
 uchar_t   hdkt_drstat;   /* drive status (ctlr specific, ro) */
};
```

`HDKIOCGBAD` The argument is a pointer to a `hdk_badmap` structure (described below). This `ioctl()` is used to get the bad sector map from the disk.

`HDKIOCSBAD` The argument is a pointer to a `hdk_badmap` structure (described below). This `ioctl()` is used to set the bad sector map on the disk.

```
/*
 * Used for bad sector map
 */
struct hdk_badmap {
    caddr_t    hdkb_bufaddr;    /* address of user's map buffer */
};
```

`HDKIOCGDIAG` The argument is a pointer to a `hdk_diag` structure (described below). This `ioctl()` gets the most recent command that failed along with the sector and error number from the hard disk.

```
/*
 * Used for disk diagnostics
 */
struct hdk_diag {
 ushort_t    hdkd_errcmd;  /* most recent command in error */
 daddr_t     hdkd_errsect; /* most recent sector in error */
 uchar_t     hdkd_errno;   /* most recent error number */
```

```
 uchar_t    hdkd_severe;  /* severity of most recent error */
};
```

See Also ioctl(2), dkio(7I)

Name heci – Intel(R) AMT Manageability Interface Driver

Description The Intel AMT Manageability Interface driver allows applications to access the Intel Active Management Technology (Intel AMT) FW by way of the host interface (as opposed to a network interface).

The Intel AMT Manageability Interface driver is meant to be used by the Local Manageability Service. When the Intel AMT machine is in Legacy Mode, the Intel AMT Manageability Interface driver functions. Messages from the Intel AMT Manageability Interface driver are sent to the system's log, that is, `/var/log/messages`.

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWamt-lms
Interface Stability	Volatile

See Also `lms(1M)`, `attributes(5)`, `e1000g(7D)`, `iwk(7D)`

Name hermon – ConnectX MT25408/MT25418/MT25428 InfiniBand (IB) Driver

Description The `hermon` driver is an IB Architecture-compliant implementation of an HCA, which operates on the Mellanox MT25408, MT25418 and MT25428 InfiniBand ASSPs using host memory for context storage rather than locally attached memory on the card. Cards based on these ASSP's utilize the PCI-Express I/O bus. These ASSP's support the link and physical layers of the InfiniBand specification while the ASSP and the driver support the transport layer.

The `hermon` driver interfaces with the InfiniBand Transport Framework (IBTF) and provides an implementation of the Channel Interfaces that are defined by that framework. It also enables management applications and agents to access the IB fabric.

Files

`/kernel/drv/hermon`
32–bit ELF kernel module. (x86)

`/kernel/drv/amd64/hermon`
64–bit ELF kernel module. (x86)

`/kernel/drv/sparcv9/hermon`
64-bit ELF Kernel Module. (SPARC)

`/kernel/drv/hermon.conf`
Driver configuration file.

Attributes See `attributes(5)` for descriptions of the following attribute:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	PCIe-based systems
Availability	SUNWhermon

See Also `driver.conf(4)`, `printers.conf(4)`, `attributes(5)`

Writing Device Drivers

Diagnostics In addition to being logged, the following messages may appear on the system console:

hermon*i*: driver attached for maintenance mode only.
There was a failure in the boot process of the hermon ASSP and the only function that can be performed is to re-flash firmware on the ASSP. (Note that *i* represents the instance of the hermon device number.)

hermon*i:* driver failed to attach
The ASSP could not boot into either operational (HCA) mode or into maintenance mode. The device is inoperable. (Note that *i* represents the instance of the hermon device number.)

Unexpected port number in port state change event.
A port state change event occurred, but the port number in the message does not exist on this HCA. This message also indicates the port number that was in the port state changed.

Hermon driver successfully detached.
The driver has been removed from the system and the HCA is no longer available for transfer operations.

hermon*i:* port *m* up.
A port up asynchronous event has occurred. (Note that *i* represents the instance of the Hermon device number while "m" represents the port number on the Hermon device.

hermon*i:* port *m* down.
A port up asynchronous event has occurred. Similar to port up event.

hermon: <command name> command failed.
A internal firmware command failed to execute.

Name hid – Human interface device (HID) class driver

Synopsis

```
keyboard@unit-address
mouse@unit-address
input@unit-address:consumer_control
#include <sys/hid.h>
int ioctl(int fildes, int command, ... /*arg*/);
```

Description The `hid` driver is a USBA (Solaris USB Architecture) compliant client driver that supports the *Human Interface Device Class (HID) 1.0* specification. The Human Interface Device (HID) class encompasses devices controlled by humans to operate computer systems. Typical examples of HID devices include keyboards, mice, trackballs, and joysticks. HID also covers front-panel controls such as knobs, switches, and buttons. A USB device with multiple interfaces may have one interface for audio and a HID interface to define the buttons that control the audio.

The `hid` driver is general and primarily handles the USB functionality of the device and generic HID functionality. For example, HID interfaces are required to have an interrupt pipe for the device to send data packets, and the `hid` driver opens the pipe to the interrupt endpoint and starts polling. The `hid` driver is also responsible for managing the device through the default control pipe. In addition to being a USB client driver, the `hid` driver is also a STREAMS driver so that modules may be pushed on top of it.

The HID specification is flexible, and HID devices dynamically describe their packets and other parameters through a HID report descriptor. The HID parser is a misc module that parses the HID report descriptor and creates a database of information about the device. The `hid` driver queries the HID parser to find out the type and characteristics of the HID device. The HID specification predefines packet formats for the boot protocol keyboard and mouse.

ioctls

`HIDIOCKMGDIRECT` This ioctl should only be addressed to a USB keyboard or mouse device. The `hid` driver maintains two streams for each USB keyboard/mouse instance: an internal one for the use of the kernel and an external one for the use of user applications. This ioctl returns the information of which stream gets the input for the moment.

arg must point to a variable of `int` type. Upon return, `0` means the internal stream gets the input, `1` means the external stream gets the input.

`HIDIOCKMSDIRECT` This ioctl should only be addressed to a USB keyboard or mouse device. The hid driver maintains two streams for each USB keyboard/mouse instance: an internal one for the use of the kernel and an external one for the use of user applications. This ioctl sets which stream should get the input for the moment.

arg must point to a variable of `int` type. The argument `0` means the internal stream gets the input, `1` means the external stream gets the input.

Files

`/kernel/drv/hid`	32–bit x86 ELF kernel hid module
`/kernel/drv/amd64/hid`	64–bit x86 ELF kernel hid module
`/kernel/drv/sparcv9/hid`	64–bit SPARC ELF kernel hid module
`/kernel/misc/hidparser`	32–bit x86 ELF kernel hidparser module
`/kernel/misc/amd64/hidparser`	64–bit x86 ELF kernel hidparser module
`/kernel/misc/sparcv9/hidparser`	64–bit SPARC ELF kernel hidparser module

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86, PCI-based systems
Availability	SUNWusb

See Also `cfgadm_usb(1M)`, `attributes(5)`, `usba(7D)`, `virtualkm(7D)`

Writing Device Drivers

STREAMS Programming Guide

Universal Serial Bus Specification 1.0 and 1.1

Device Class Definition for Human Interface Devices (HID) 1.1

System Administration Guide: Basic Administration

http://www.sun.com

Diagnostics

hid_attach: Unsupported HID device.	The device requires a protocol not supported by the hid driver.
Parsing of hid descriptor failed.	The HID report descriptor cannot be parsed correctly. The device cannot be supported by the hid driver.
Invalid report descriptor.	The HID report descriptor is invalid. The device cannot be supported by the hid driver.

The following messages may be logged into the system log. They are formatted in the following manner:

```
<device path><hid<instance number>): message...
```

hid_attach: Unsupported HID device.	The device cannot be supported by this version of the HID driver.

Parsing of HID descriptor failed.	The device cannot be supported by this version of the HID driver.
Invalid report descriptor.	The device cannot be supported by this version of the HID driver.

Notes The hid driver currently supports only keyboard, mouse and audio HID control devices.

Normally a mouse is not power managed and consquently, screen darkening can be undone with a mouse movement. If power management of the mouse is required, add the following line to `hid.conf` then reboot the system:

```
hid-mouse-pm-enable;
```

Modern mice that are power managed require a 'click' to wake up. Occasionally, this may cause unexpected results.

Name hme – SUNW,hme Fast-Ethernet device driver

Synopsis `/dev/hme`

Description The `SUNW,hme` Fast-Ethernet driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface, `dlpi(7P)`, over a `SUNW,hme` Fast-Ethernet controller. The motherboard and add-in SBus `SUNW,hme` controllers of several varieties are supported. Multiple `SUNW,hme` controllers installed within the system are supported by the driver.

The `hme` driver provides basic support for the `SUNW,hme` hardware. It is used to handle the `SUNW,hme` device. Functions include chip initialization, frame transit and receive, multicast and promiscuous support, and error recovery and reporting. `SUNW,hme` The `SUNW,hme` device provides 100Base-TX networking interfaces using SUN's `FEPS ASIC` and an Internal Transceiver. The FEPS ASIC provides the Sbus interface and MAC functions and the Physical layer functions are provided by the Internal Transceiver which connects to a `RJ-45` connector. In addition to the RJ-45 connector, an `MII` (Media Independent Interface) connector is also provided on all `SUNW,hme` devices except the SunSwith SBus adapter board. The MII interface is used to connect to an External Transceiver which may use any physical media (copper or fiber) specified in the 100Base-TX standard. When an External Transceiver is connected to the MII, the driver selects the External Transceiver and disables the Internal Transceiver.

The 100Base-TX standard specifies an “auto-negotiation” protocol to automatically select the mode and speed of operation. The Internal transceiver is capable of doing “auto-negotiation” with the remote-end of the link (Link Partner) and receives the capabilities of the remote end. It selects the `Highest Common Denominator` mode of operation based on the priorities. It also supports `forced-mode` of operation where the driver can select the mode of operation.

Application Programming Interface The cloning character-special device `/dev/hme` is used to access all `SUNW,hme` controllers installed within the system.

hme and DLPI The `hme` driver is a “style 2” Data Link Service provider. All `M_PROTO` and `M_PCPROTO` type messages are interpreted as `DLPI` primitives. Valid `DLPI` primitives are defined in `<sys/dlpi.h>`. Refer to `dlpi(7P)` for more information. An explicit `DL_ATTACH_REQ` message by the user is required to associate the opened stream with a particular device (`ppa`). The `ppa` ID is interpreted as an `unsigned long` data type and indicates the corresponding device instance (unit) number. An error (`DL_ERROR_ACK`) is returned by the driver if the `ppa` field value does not correspond to a valid device instance number for this system. The device is initialized on first attach and de-initialized (stopped) at last detach.

The values returned by the driver in the `DL_INFO_ACK` primitive in response to the `DL_INFO_REQ` from the user are as follows:

- The maximum SDU is `1500` (`ETHERMTU` - defined in <sys/ethernet.h>).
- The minimum SDU is `0`.

- The dlsap address length is 8.
- The MAC type is DL_ETHER.
- The sap length values is –2 meaning the physical address component is followed immediately by a 2 byte sap component within the DLSAP address.
- The service mode is DL_CLDLS.
- No optional quality of service (QOS) support is included at present so the QOS fields are 0.
- The provider style is DL_STYLE2.
- The version is DL_VERSION_2.
- The broadcast address value is Ethernet/IEEE broadcast address (0xFFFFFF).

Once in the DL_ATTACHED state, the user must send a DL_BIND_REQ to associate a particular SAP (Service Access Pointer) with the stream. The hme driver interprets the sap field within the DL_BIND_REQ as an Ethernet "type" therefore valid values for the sap field are in the [0-0xFFFF] range. Only one Ethernet type can be bound to the stream at any time.

If the user selects a sap with a value of 0, the receiver will be in "802.3 mode". All frames received from the media having a "type" field in the range [0-1500] are assumed to be 802.3 frames and are routed up all open Streams which are bound to sap value 0. If more than one Stream is in "802.3 mode" then the frame will be duplicated and routed up multiple Streams as DL_UNITDATA_IND messages.

In transmission, the driver checks the sap field of the DL_BIND_REQ if the sap value is 0, and if the destination type field is in the range [0-1500]. If either is true, the driver computes the length of the message, not including initial M_PROTO mblk (message block), of all subsequent DL_UNITDATA_REQ messages and transmits 802.3 frames that have this value in the MAC frame header length field.

The hme driver DLSAP address format consists of the 6 byte physical (Ethernet) address component followed immediately by the 2 byte sap (type) component producing an 8 byte DLSAP address. Applications should *not* hardcode to this particular implementation-specific DLSAP address format but use information returned in the DL_INFO_ACK primitive to compose and decompose DLSAP addresses. The sap length, full DLSAP length, and sap/physical ordering are included within the DL_INFO_ACK. The physical address length can be computed by subtracting the sap length from the full DLSAP address length or by issuing the DL_PHYS_ADDR_REQ to obtain the current physical address associated with the stream.

Once in the DL_BOUND state, the user may transmit frames on the Ethernet by sending DL_UNITDATA_REQ messages to the hme driver. The hme driver will route received Ethernet frames up all those open and bound streams having a sap which matches the Ethernet type as DL_UNITDATA_IND messages. Received Ethernet frames are duplicated and routed up multiple

open streams if necessary. The DLSAP address contained within the DL_UNITDATA_REQ and DL_UNITDATA_IND messages consists of both the sap (type) and physical (Ethernet) components.

In addition to the mandatory connectionless DLPI message set the driver additionally supports the following primitives.

hme Primitives

The DL_ENABMULTI_REQ and DL_DISABMULTI_REQ primitives enable/disable reception of individual multicast group addresses. A set of multicast addresses may be iteratively created and modified on a per-stream basis using these primitives. These primitives are accepted by the driver in any state following DL_ATTACHED.

The DL_PROMISCON_REQ and DL_PROMISCOFF_REQ primitives with the DL_PROMISC_PHYS flag set in the dl_level field enables/disables reception of all ("promiscuous mode") frames on the media including frames generated by the local host. When used with the DL_PROMISC_SAP flag set this enables/disables reception of all sap (Ethernet type) values. When used with the DL_PROMISC_MULTI flag set this enables/disables reception of all multicast group addresses. The effect of each is always on a per-stream basis and independent of the other sap and physical level configurations on this stream or other streams.

The DL_PHYS_ADDR_REQ primitive returns the 6 octet Ethernet address currently associated (attached) to the stream in the DL_PHYS_ADDR_ACK primitive. This primitive is valid only in states following a successful DL_ATTACH_REQ.

The DL_SET_PHYS_ADDR_REQ primitive changes the 6 octet Ethernet address currently associated (attached) to this stream. The credentials of the process which originally opened this stream must be superuser. Otherwise EPERM is returned in the DL_ERROR_ACK. This primitive is destructive in that it affects all other current and future streams attached to this device. An M_ERROR is sent up all other streams attached to this device when this primitive is successful on this stream. Once changed, all streams subsequently opened and attached to this device will obtain this new physical address. Once changed, the physical address will remain until this primitive is used to change the physical address again or the system is rebooted, whichever comes first.

hme DRIVER

By default, the hme driver performs "auto-negotiation" to select the mode and speed of the link, when the Internal Transceiver is used.

When an External Transceiver is connected to the MII interface, the driver selects the External Transceiver for networking operations. If the External Transceiver supports "auto-negotiation", the driver uses the auto-negotiation procedure to select the link speed and mode. If the External Transceiver does not support auto-negotiation, it will select the highest priority mode supported by the transceiver.

- 100 Mbps, full-duplex
- 100 Mbps, half-duplex
- 10 Mbps, full-duplex

- 10 Mbps, half-duplex

The link can be in one of the *4* following modes:

These speeds and modes are described in the 100Base-TX standard.

The *auto–negotiation* protocol automatically selects:

- Operation mode (half-duplex or full-duplex)
- Speed (100 Mbps or 10 Mbps)

The auto–negotiation protocol does the following:

- Gets all the modes of operation supported by the Link Partner
- Advertises its capabilities to the Link Partner
- Selects the highest common denominator mode of operation based on the priorities

The *internal transceiver* is capable of all of the operating speeds and modes listed above. When the internal transceiver is used, by *default*, auto-negotiation is used to select the speed and the mode of the link and the common mode of operation with the Link Partner.

When an *external transceiver* is connected to the `MII` interface, the driver selects the external transceiver for networking operations. If the external transceiver supports auto-negotiation:

- The driver uses the auto-negotiation procedure to select the link speed and mode.

If the external transceiver *does not* support auto-negotiation

- The driver selects the highest priority mode supported by the transceiver.

Sometimes, the user may want to select the speed and mode of the link. The `SUNW,hme` device supports programmable `"IPG"` (Inter-Packet Gap) parameters `ipg1` and `ipg2`. By default, the driver sets `ipg1` to 8 `byte-times` and `ipg2` to 4 `byte-times` (which are the standard values). Sometimes, the user may want to alter these values depending on whether the driver supports 10 Mbps or 100 Mpbs and accordingly, `IPG` will be set to 9.6 or 0.96 microseconds.

hme Parameter List

The hme driver provides for setting and getting various parameters for the `SUNW,hme` device. The parameter list includes:

```
current transceiver status
current link status
inter-packet gap
local transceiver capabilities
link partner capabilities
```

The local transceiver has two set of capabilities: one set reflects the capabilities of the `hardware`, which are `read-only (RO)` parameters and the second set reflects the values chosen

by the user and is used in `speed selection`. There are `read/write (RW)` capabilities. At boot time, these two sets of capabilities will be the same. The Link Partner capabilities are also read only parameters because the current default value of these parameters can only be read and cannot be modified.

Files

`/dev/hme`	`hme` special character device
`/kernel/drv/hme.conf`	System-wide default device driver properties

See Also `ndd(1M)`, `netstat(1M)`, `driver.conf(4)`, `dlpi(7P)`

Name hpfc – Agilent fibre channel host bus adapter

Synopsis `PCI pci103c`

Description The `hpfc` fibre channel host bus adapter is a SCSA compliant nexus driver that supports all Agilent fibre channel host bus adapters, including the HHBA5100*x*, HHBA5101*x*, and HHBA5121*x* models. Agilent host bus adapters support the fibre channel protocol on private fibre channel arbitrated loops and fabrics. The driver supports up to ten host bus adapters, with a maximum of 125 fibre channel devices on each host bus adapter. The `hpfc` driver supports a maximum of 256 LUNs per target.

The `hpfc` driver does not support the BIOS Int 13 feature, which enables the booting of an operating system. As a result, you should not install an operating system on devices attached to the `hpfc` driver.

Configuration The `hpfc` driver attempts to configure itself using the information in the `/kernel/drv/hpfc.conf` configuration file.

By default, the driver supports only LUN 0 for each target device. To add multiple LUN support, modify the `/kernel/drv/sd.conf` file.

Before upgrading the `hpfc` driver, backup the `sd.conf` file to save customized LUN settings and then use `pkgrm(1M)` to remove the old version of the driver.

The host bus adapter port is initialized to `FL_Port` when connected to a fabric switch. To change it to `F_Port`, add the `init_as_nport=1` entry to the `hpfc.conf` file and reboot the system.

To conserve system resources, at least one disk drive must be attached to the `hpfc` driver. If no devices are attached, the driver will not load.

Files

`/kernel/drv/hpfc`	32–bit ELF kernel module
`/kernel/drv/sparcv9/hpfc`	64–bit ELF kernel module
`/kernel/drv/hpfc.conf`	Driver configuration file
`/kernel/drv/sd.conf`	SCSI disk configuration file

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86, SPARC

See Also `luxadm(1M)`, `pkgrm(1M)`, `prtconf(1M)`, `driver.conf(4)`, `attributes(5)`, `ses(7D)`, `ssd(7D)`

ANSI X3.272–1996, Fibre Channel Arbitrated Loop (FC-AL),

ANSI X3.269-1996, Fibre Channel Protocol for SCSI (FCP),

ANSI X3.270-1996, SCSI-3 Architecture Model (SAM),

Fibre Channel Private Loop SCSI Direct Attach (FC-PLDA)

Name hsfs – High Sierra & ISO 9660 CD-ROM file system

Description HSFS is a file system type that allows users to access files on High Sierra or ISO 9660 format CD-ROM disks from within the SunOS operating system. Once mounted, a HSFS file system provides standard SunOS read-only file system operations and semantics, meaning that you can read and list files in a directory on a High Sierra or ISO 9660 CD-ROM and applications can use standard UNIX system calls on these files and directories.

This file system contains support for Rock Ridge, ISO 9660 Version 2 and Joliet extensions. These extensions provide support for file names with a length of at least 207 bytes, but only Rock Ridge extensions (with the exception of writability and hard links) can provide file system semantics and file types as they are found in UFS. The presence of Rock Ridge, ISO 9660 Version 2 and Joliet is autodetected and the best-suitable available extension is used by the HSFS driver for file name and attribute lookup.

If your `/etc/vfstab` file contains a line similar to the following:

```
/dev/dsk/c0t6d0s0 –/hsfs hsfs -no ro
```

and `/hsfs` exists, you can mount an HSFS file system with either of the following commands:

`mount -F hsfs -o ro` *device-special directory-name*

or

```
mount /hsfs
```

By default, Rock Ridge extensions are used if available, otherwise ISO 9660 Version 2, then Joliet are used. If neither extension is present HSFS defaults to the standard capabilities of ISO 9660. Since so-called hybrid CD-ROMs that contain multiple extensions are possible, you can use the following mount options to deliberately disable the search for a specific extension or to force the use of a specific extension even if a preferable type is present:

`mount -F hsfs -o ro,nrr` *device-special directory-name*

Mount options are:

rr—request HSFS to use Rock Ridge extensions, if present. This is the default behavior and does not need to be explicitly specified.

nrr—disable detection and use of Rock Ridge extensions, even if present.

vers2—request HSFS to use *ISO 9660* Version 2 extensions, even if Rock Ridge is available.

novers2—disable detection and use of *ISO 9660* Version 2 extensions.

joliet—request HSFS to use Joliet extensions, even if Rock Ridge or *ISO 9660* Version 2 extensions are available.

nojoliet—disable detection and use of Joliet extensions.

Files on a High Sierra or ISO 9660 CD-ROM disk have names of the form *filename.ext;version*, where *filename* and the optional *ext* consist of a sequence of uppercase alphanumeric characters (including "_"), while the *version* consists of a sequence of digits, representing the version number of the file. HSFS converts all the uppercase characters in a file name to lowercase, and truncates the ";" and version information. If more than one version of a file is present on the CD-ROM, only the file with the highest version number is accessible.

Conversion of uppercase to lowercase characters may be disabled by using the `-o nomaplcase` option to `mount(1M)`. (See `mount_hsfs(1M)`).

If the CD-ROM contains Rock Ridge, ISO 9660 version 2 or Joliet extensions, the file names and directory names may contain any character supported under UFS. The names may also be upper and/or lower case and are case sensitive. File name lengths can be as long as those of UFS.

Files accessed through HSFS have mode 555 (owner, group and world readable and executable), uid 0 and gid 3. If a directory on the CD-ROM has read permission, HSFS grants execute permission to the directory, allowing it to be searched.

With Rock Ridge extensions, files and directories can have any permissions that are supported on a UFS file system. However, under all write permissions, the file system is read-only, with `EROFS` returned to any write operations.

Like High Sierra and ISO 9660 CD-ROMs, HSFS supports only regular files and directories. A Rock Ridge CD-ROM can support regular files, directories, and symbolic links, as well as device nodes, such as block, character, and FIFO.

Examples

EXAMPLE 1 Sample Display of File System Files

If there is a file `BIG.BAR` on a High Sierra or ISO 9660 format CD-ROM it will show up as `big.bar` when listed on a HSFS file system.

If there are three files

```
BAR.BAZ;1
```

```
BAR.BAZ;2
```

and

```
BAR.BAZ;3
```

on a High Sierra or ISO 9660 format CD-ROM, only the file `BAR.BAZ;3` will be accessible. It will be listed as `bar.baz`.

See Also

`mount(1M)`, `mount_hsfs(1M)`, `vfstab(4)`

N. V. Phillips and Sony Corporation, *System Description Compact Disc Digital Audio*, ("Red Book").

N. V. Phillips and Sony Corporation, *System Description of Compact Disc Read Only Memory*, ("Yellow Book").

IR "Volume and File Structure of CD-ROM for Information Interchange", ISO 9660:1988(E).

Diagnostics

hsfs: Warning: the file system... does not conform to the ISO-9660 spec	The specific reason appears on the following line. You might be attempting to mount a CD-ROM containing a different file system, such as UFS.
hsfs: Warning: the file system... contains a file [with an] unsupported type	The `hsfs` file system does not support the format of some file or directory on the CD-ROM, for example a record structured file.
hsfs: hsnode table full, %d nodes allocated	There are not enough HSFS internal data structure elements to handle all the files currently open. This problem may be overcome by adding a line of the form `set hsfs:nhsnode=`*number* to the `/etc/system` system configuration file and rebooting. See `system(4)`.

Warnings

Do not physically eject a CD-ROM while the device is still mounted as a HSFS file system.

Under MS-DOS (for which CD-ROMs are frequently targeted), files with no extension may be represented either as:

filename.

or

filename

that is, with or without a trailing period. These names are not equivalent under UNIX systems. For example, the names:

`BAR.`

and

`BAR`

are not names for the same file under the UNIX system. This may cause confusion if you are consulting documentation for CD-ROMs originally intended for MS-DOS systems.

Use of the `-o notraildot` option to `mount(1M)` makes it optional to specify the trailing dot. (See `mount_hsfs(1M)`).

Notes No translation of any sort is done on the contents of High Sierra or ISO 9660 format CD-ROMs; only directory and file names are subject to interpretation by HSFS.

Name hubd – USB hub driver

Synopsis `hub@unit-address`

Description The `hubd` is a USBA (Solaris USB Architecture) compliant client driver that supports USB hubs conforming to the *Universal Serial Bus Specification 2.0*. The `hubd` driver supports bus–powered and self–powered hubs. The driver supports hubs with individual port power, ganged power and no power switching.

When a device is attached to a hub port, the `hubd` driver enumerates the device by determining its type and assigning an address to it. For multi-configuration devices, `hubd` sets the preferred configuration (refer to `cfgadm_usb`(1M) to select a configuration). The `hubd` driver attaches a driver to the device if one is available for the default or selected configuration. When the device is disconnected from the hub port, the `hubd` driver offlines any driver instance attached to the device.

Files

`/kernel/drv/hubd`	32– bit x86 ELF kernel module
`/kernel/drv/amd64/hubd`	64– bit x86 ELF kernel module
`/kernel/drv/sparcv9/hubd`	64–bit SPARC ELF kernel module

Attributes See `attributes`(5) for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86, PCI-based systems
Availability	SUNWusb

See Also `cfgadm_usb`(1M), `attributes`(5), `usba`(7D)

Writing Device Drivers

Universal Serial Bus Specification 2.0

System Administration Guide: Basic Administration

http://*www.sun.com/io*

Diagnostics In addition to being logged, the following messages may also appear on the system console. Messages are formatted in the following manner:

```
WARNING: <device path> <hubd<instance number>): Message...
```

where `<instance number>` is the instance number of `hubd` and `<device path>` is the physical path to the device in `/devices` directory. Messages from the root hub are displayed with a `usb<instance number>` prefix instead of `hub<instance number>` as the root hub is an integrated part of the host controller.

Connecting device on port <*number*> failed.

The driver failed to enumerate the device connected on port <*number*> of hub. If enumeration fails, disconnect and re-connect.

Use of a USB 1.0 hub behind a high speed port may cause unexpected failures.

Devices connected to a USB 1.0 hub which are in turn connected to an external USB 2.0 hub, may misbehave unexpectedly or suddenly go offline. This is due to a documented incompatibility between USB 1.0 hubs and USB 2.0 hub Transaction Translators. Please use only USB 2.0 or USB 1.1 hubs behind high-speed ports.

Connecting a high speed device to a non-high speed hub (port x) will result in a loss of performance. Please connect the device to a high speed port to get the maximum performance.

USB 2.0 devices connected to USB 1.0 or 1.1 hubs cannot run at their highest speed, even when the hub is in turn connected to a high-speed port. For best performance, reconnect without going through a USB 1.0 or 1.1 hub.

Cannot access <device>. Please reconnect.

This hub has been disconnected because a device other than the original one has been inserted. The driver informs you of this fact by displaying the name of the original device.

Port <n> overcurrent.

An overcurrent condition was detected. Please remove the device on this port.

Devices not identical to the previous one on this port. Please disconnect and reconnect.

Same condition as described above; however in this case, the driver is unable to identify the original device with a name string.

Hub driver supports max of <*n*> ports on hub. Hence, using the first <*number of physical ports*> of <*n*> ports available.

The current hub driver supports hubs that have <*n*> ports or less. A hub with more than <*n*> ports has been plugged in. Only the first <*n*> out of the total <*number of physical ports*> ports are usable.

Hub global over current condition, please disconnect the devices connected to the hub to clear the condition. You may need to re-connect the hub if the ports do not work.

An overcurrent condition was detected on the hub. This means that the aggregate current being drawn by the devices on the downstream ports exceeds a preset value. Refer to section 7.2.1.2 and 11.13 of the *Universal Serial Bus Specification 2.0*. If this message continues to display, you may need to remove downstream devices to eliminate the problem. If any port does not work after the overcurrent condition is cleared, re-connect the hub to re-enable the ports.

Root hub over current condition, please check your system to clear the condition as soon as possible. You may need to reboot the system if the root hub does not recover automatically.

An overcurrent condition was detected on the root hub, indicating that malfunctioning devices on the downstream ports are drawing too much current. Please disconnect the problematic downstream devices to eliminate the problem. If the root hub doesn't work

after the overcurrent condition is cleared, you may need to reboot the system.

The following messages may be logged into the system log. They are formatted in the following manner:

```
<device path><hubd<instance number>): message...
```

Local power has been lost, please disconnect hub.

A USB self-powered hub has lost external power. All USB devices connected down-stream from this hub will cease to function. Disconnect the hub, plug in the external power-supply and then plug in the hub again.

Local power has been lost, the hub could draw <x> mA power from the USB bus.

A USB self/bus-powered hub has lost external power. Some USB devices connected down-stream from this hub may cease to function. Disconnect the external power-supply and then plug in the hub again.

Two bus-powered hubs cannot be concatenated.

A bus-powered hub was connected to a bus powered hub port. Please remove this bus-powered hub and connect it to a self-powered hub or a root hub port.

Configuration <n> for device <device> at port <m> exceeds power available for this port. Please re-insert your device into another hub port which has enough power.

The device requires more power than is available on this port.

Port <*n*> in over current condition, please check the attached device to clear the condition. The system will try to recover the port, but if not successful, you need to re-connect the hub or reboot the system to bring the port back to work.

An overcurrent condition was detected on port <*n*>. This means the device connected to the port is drawing more current than the hub can supply. If this message continues to display, please disconnect the device to eliminate the problem. If the port doesn't work after the overcurrent condition is cleared, please re-connect the hub or reboot the system to enable the port again.

Name hwahc – Host Wire Adapter Host Controller Driver

Description The `hwahc` driver is a USBA (Solaris USB Architecture) compliant nexus driver that supports the Wireless USB 1.0 Host Wire Adapter Host Controller, an industry standard developed by USB-IF.

A Host Wire Adapter (HWA) is a USB device whose upstream connection is a USB 2.0 wired interface. The HWA operates as a host to a cluster of downstream Wireless USB devices.

The `hwahc` driver supports bulk, interrupt and control transfers.

Files

`/kernel/drv/hwahc`	32-bit ELF 86 kernel module
`/kernel/drv/sparcv9/hwahc`	64-bit SPARC ELF kernel module
`/kernel/drv/amd64/hwahc`	64-bit x86 ELF kernel module

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86
Availability	SUNWusb

See Also `add_drv(1M)`, `prtconf(1M)`, `rem_drv(1M)`, `update_drv(1M)`, `attributes(5)`, `ehci(7D)`, `hubd(7D)`, `usba(7D)`

Writing Device Drivers

System Administration Guide: Basic Administration

Universal Serial Bus Specification 2.0

Wireless Universal Serial Bus Specification 1.0

http://www.usb.org

http://www.sun.com

Diagnostics All host controller errors are passed to the client drivers. In addition to being logged, the following messages can appear on the system console. All messages are formatted in the following way:

`WARNING:` *device_path hwahc instance_number*`: Message ...`

`Connection device on WUSB port` *port_number* `fails`

The connecting device fails to connect to the HWA. Make sure the device has been associated with the host.

Name hwarc – HWA Radio Controller Driver

Synopsis `hwa-radio@unit-address`

Description The `hwarc` driver is a USBA (Solaris USB Architecture) compliant client driver that supports Host Wire Adapter Radio Controller, specified in Wireless Universal Serial Bus Specification, Version 1.0.

The `hwarc` driver handles the Radio Controller Interface of an HWA device and properly controls the UWB (Ultra Wideband) Radio in the device. The driver controls an HWA device to Scan, Start/Stop Beacon, Get IE, and so forth.

Files

`/kernel/drv/hwarc`	32-bit ELF 86 kernel module
`/kernel/drv/sparcv9/hwarc`	64-bit SPARC ELF kernel module
`/kernel/drv/amd64/hwarc`	64-bit x86 ELF kernel module

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86
Availability	SUNWusb

See Also `add_drv(1M)`, `prtconf(1M)`, `rem_drv(1M)`, `update_drv(1M)`, `attributes(5)`, `hwahc(7D)`, `usba(7D)`, `uwba(7D)`,

Writing Device Drivers

Universal Serial Bus Specification 1.0, 1.1 and 2.0 - 1996, 1998, 2000

Wireless Universal Serial Bus Specification 1.0

http://www.usb.org

http://www.sun.com

Name hxge – Sun Blade 10 Gigabit Ethernet network driver

Synopsis `/dev/hxge*`

Description The `hxge` Gigabit Ethernet driver is a multi-threaded, loadable, clonable, GLD-based STREAMS driver supporting the Data Link Provider Interface, `dlpi(7P)`, on the Sun Blade Shared 10Gb Ethernet Interface.

The Shared PCI-Express 10 Gb networking interface provides network I/O consolidation for up to six Constellation blades, with each blade seeing its own portion of the network interface.

The `hxge` driver functions include chip initialization, frame transmit and receive, flow classification, multicast and promiscuous support and error recovery and reporting in the blade domain.

Application Programming Interface The cloning character-special device, `/dev/hxge`, is used to access Sun Blade Shared 10Gb Ethernet Interface devices installed within the system.

The `hxge` driver is managed by the `dladm(1M)` command line utility, which allows VLANs to be defined on top of `hxge` instances and for `hxge` instances to be aggregated. See `dladm(1M)` for more details.

You must send an explicit DL_ATTACH_REQ message to associate the opened stream with a particular device (PPA). The PPA ID is interpreted as an unsigned integer data type and indicates the corresponding device instance (unit) number. The driver returns an error (DL_ERROR_ACK) if the PPA field value does not correspond to a valid device instance number for the system. The device is initialized on first attach and de-initialized (stopped) at last detach

The values returned by the driver in the DL_INFO_ACK primitive in response to a DL_INFO_REQ are:

- Maximum SDU is 1500 (ETHERMTU - defined in <`sys/ethernet.h`>).
- Minimum SDU is 0.
- DLSAP address length is 8.
- MAC type is `DL_ETHER`.
- `SAP` length value is *-2*, meaning the physical address component is followed immediately by a 2-byte `SAP` component within the `DLSAP` address.
- Broadcast address value is the Ethernet/IEEE broadcast address (FF:FF:FF:FF:FF:FF).

  Due to the nature of the link address definition for IPoIB, the DL_SET_PHYS_ADDR_REQ DLPI primitive is not supported.

  In the transmit case for streams that have been put in raw mode via the DLIOCRAW ioctl, the `dlpi` application must prepend the 20 byte IPoIB destination address to the data it wants to transmit over-the-wire. In the receive case, applications receive the IP/ARP datagram along with the IETF defined 4 byte header.

Once in the DL_ATTACHED state, you must send a DL_BIND_REQ to associate a particular Service Access Point (SAP) with the stream.

Configuration The link speed and mode are fixed at 10 Gbps full-duplex.

The default MTU is 1500. To enable jumbo frame support, you configure the `hxge` driver by defining the accept-jumbo property to 1 in the `hxge.conf` file. Note that the largest jumbo size is 9178 bytes.

The driver may be configured to discard certain classes of traffic. By default, no class of traffic is allowed. You configure the `hxge` driver by defining the class option property to 0x20000 in `hxge.conf` to discard the specified class of traffic. For example, the following line in `hxge.conf` discards all IP Version 4 TCP traffic:

```
class-opt-ipv4-tcp = 0x20000;
```

You can also use the `ndd(1M)` command to configure the `hxge` driver at runtime to discard any classes of traffic.

The `hxge`driver supports the self-healing functionality of Solaris OS. By default it is configured to DDI_FM_EREPORT_CAPABLE | DDI_FM_ERRCB_CAPABLE. You configure the `hxge` driver by defining the fm-capable property in `hxge.conf` to other capabilities or to 0x0 to disable it entirely.

The `hxge` driver may be configured using the standard `ifconfig(1M)` command.

The `hxge` driver also reports various hardware and software statistics data. You can view these statistics using the `kstat(1M)` command.

Files

`/dev/hxge*`	Special character device.
`/kernel/drv/hxge`	32–bit device driver (x86).
`/kernel/drv/sparcv9/hxge`	64–bit device driver (SPARC).
`/kernel/drv/amd64/hxge`	64–bit device driver (x86).
`/kernel/drv/hxge.conf`	Configuration file.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86

See Also `dladm(1M)`, `ifconfig(1M)`, `kstat(1M)`, `ndd(1M)`, `netstat(1M)`, `driver.conf(4)`, `attributes(5)`, `streamio(7I)`, `dlpi(7P)`

Writing Device Drivers

STREAMS Programming Guide

Network Interfaces Programmer's Guide

Name i915 – DRI-compliant kernel driver providing graphic hardware acceleration support

Description The `i915` driver is a Direct Rendering Infrastructure (DRI)– compliant kernel driver that provides graphics hardware acceleration support. DRI is a framework for coordinating OS kernel, 3D graphics hardware, X window system and OpenGL applications.

The `i915` driver currently supports the Intel i845, i865, i915, i945, i965 and G33 series integrated graphics controllers.

Files `/platform/i86pc/kernel/drv/i915` 32–bit ELF kernel module (x86).

`/platform/i86pc/kernel/drv/amd64/i915` 64–bit ELF kernel module (x86).

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWdrmr
Architecture	x86

See Also `attributes(5)`

`/usr/X11/share/man/man1/Xserver.1`

`/usr/X11/share/man/man1/Xorg.1`

`/usr/X11/share/man/man5/X11.5`

Name ib – InfiniBand Bus Nexus Driver

Description The `ib` (IB nexus) driver is a pseudo nexus driver that supports enumeration of port devices, VPPA (Virtual Physical Point Attachment), HCA_SVC (HCA Service) devices, and I/O controllers (IOC) on the InfiniBand fabric that are visible to the host and provides interfaces to `cfgadm_ib(1M)` to manage hot-plugging of IB devices. The `ib` nexus driver enumerates the port device, VPPA devices and HCA_SVC devices based on entries specified in the `ib.conf` file. IOC devices are enumerated on demand. The IB nexus driver uses InfiniBand Device Manager services (`ibdm(7D)`) to enumerate port devices, VPPA devices, HCA_SVC devices, and IOCs on the IB fabric.

Configuration You configure the `ib` driver by defining properties in the `ib.conf` file. The IB nexus driver supports the following properties:

PROPERTY NAME	DEFAULT	POSSIBLE VALUES
port-svc-list	""	List of service names, for example: srv
vppa-svc-list	""	List of service names, for example: ipib
hca-svc-list	""	List of service names, for example: hca_nfs

The `port-svc-list` property defines the list of port communication service names per port. The IB nexus driver creates a device instance for each entry in this property per Host Channel Adapter (HCA) port. The `ib.conf` file contains a `port-svc-list=""` entry by default. You update `port-svc-list` with service names you want to add to the system.

The `vppa-svc-list` property defines the list of VPPA communication service names per port per partition key. The IB nexus driver creates a device instance for each entry in this property per Host Channel Adapter (HCA) port. The `ib.conf` file contains a `vppa-svc-list=""` entry by default. You update `vppa-svc-list` with service names you want to add to the system.

The `hca-svc-list` property defines the list of HCA_SVC communication service names per HCA. The IB nexus driver creates a device instance for each entry in this property per Host Channel Adapter (HCA). The `ib.conf` file contains a `hca-svc-list=""` entry by default. You update `hca-svc-list` with service names you want to add to the system.

The service name specified in `port-svc-list`, `vppa-svc-list` and `hca-svc-list` must be unique, be a maximum of four characters long, and is limited to digits 0-9 and letters a-z and A-Z.

IOC drivers (which are parented by the IB nexus driver) may themselves have .conf files. To distinguish those cases from pseudo drivers parented by IB nexus, such drivers should include

the "ib-node-type" property with value merge in the IOC driver.conf file. That property ensures that properties from the .conf file are merged with other properties found through hardware probing.

Examples Example 1: A sample ib.conf file with one service name entry for PORT communication services.

```
#
# Copyright 2001-2003 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
port-svc-list=""
vppa-svc-list="";
hca-svc-list="";
```

In Example 1, the IB nexus driver does not create any port/vppa/hca_svc device instances.

Example 2: A sample ib.conf file with one entry for "srv" service:

```
port-svc-list="srv"
vppa-svc-list="";
hca-svc-list="";
```

The IB nexus driver creates one srv service instance for every HCA port that exists on the host. For example, if there are two HCAs, each with two ports on the host, the IB nexus driver creates four instances of the srv service.

Example 3: A sample ib.conf file with one service name entry for each of Port and VPPA communication services:

```
port-svc-list="srv"
vppa-svc-list="ipib";
hca-svc-list="";
```

If there are two HCAs in the system with two ports each and each port has two valid PKEY values, the IB nexus driver creates four instances of srv service (one for each port). It also creates eight instances of ipd service (one per each port/PKEY combination).

Example 4: A sample ib.conf file with one service name entry for each of Port, VPPA and HCA_SVC communication services:

```
port-svc-list="srv";
vppa-svc-list="ipib";
hca-svc-list="hca_nfs";
```

The IB nexus driver creates one instance of hca_nfs service for each HCA in the system.

Example 5: IOC driver .conf

```
ib-node-type="merge";
enable-special-mode="on";
```

Files

`/kernel/drv/ib`	32-bit x86 ELF kernel module
`/kernel/drv/amd64/ib`	64-bit x86 ELF kernel module
`/kernel/drv/sparcv9/ib`	64-bit SPARC ELF kernel module
`/kernel/drv/ib.conf`	driver configuration file

Attributes See `attributes(5)` for a description of the following attribute:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86
Availability	SUNWhea, SUNWib
Interface Stability	Consolidation Private

See Also `cfgadm_ib(1M)`, `driver.conf(4)`, `ib(4)`, `attributes(5)`, `ibcm(7D)`, `ibdm(7D)`, `ibtl(7D)`

Writing Device Drivers

InfiniBand Architecture Specification, Volume 1: Release 1.1

System Administration Guide: Basic Administration

Diagnostics In addition to being logged, the following messages may appear on the system console. All messages are formatted in the following manner:

```
ib: WARNING: Error message...
```

unit-address property in `%s.conf` not well-formed.	The `driver.conf` file does not have a valid "`unit-addr`" property defined. This property is an array of strings.
cannot find unit-address in `%s.conf`.	The `driver.conf` file does not have a valid "`unit-addr`" property defined. This property is an array of strings.
Waiting for Port %d initialization.	Waiting for port initialization from subnet manager.

Name ibcm – Solaris InfiniBand Communication Manager

Description The Solaris InfiniBand Communication Manager (IBCM) is a Solaris kernel misc module that adheres to the *InfiniBand Architecture Specification, Volume 1: Release 1.1* for InfiniBand Communication Management Class.

IBCM provides a transport layer abstraction to IB clients to set up reliable connected channels along with service, multicast, and path lookup-related functionality. IBCM implements the CM protocol as per the *InfiniBand Architecture Specification, Volume 1: Release 1.1* and utilizes the InfiniBand Management Framework module for all IB management-related functionality and the InfiniBand Transport Layer (see `ibtl(7D)`) for all IB Verbs-related functionality.

Files

`/kernel/misc/ibcm`	32-bit x86 ELF kernel module
`/kernel/misc/amd64/ibcm`	64-bit x86 ELF kernel module
`/kernel/misc/sparcv9/ibcm`	64-bit SPARC ELF kernel module

Attributes See `attributes(5)` for a description of the following attribute:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Consolidation Private
Availability	SUNWhea

See Also `attributes(5)`, `ibtl(7D)`

InfiniBand Architecture Specification, Volume 1: Release 1.1

Name ibd – Infiniband IPoIB device driver

Synopsis `/dev/ibd*`

Description The `ibd` driver implements the IETF IP over Infiniband protocol and provides IPoIB service for all IBA ports present in the system.

The `ibd` driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface, `dlpi(7P)`).

By default, datagram mode is used by each `ibd` instance, unless the `enable_rc` is set to `1` for that instance in the .conf file. This change can be made on a per instance basis by changing the corresponding value of the variable. So the *N*`th` value of `enable_rc` changes the setting for the *N*`th` instance of `ibd`. Any value other than `1`, or no `.conf` file at all is equivalent to specifying datagram mode.

Because `ibd` over connected mode attempts to use a large MTU (65520 bytes), applications should adapt to the large MTU to get better performance, for example, adopting a large TCP window size.

Use the cloning, character-special device `/dev/ibd` to access all `ibd` devices installed within the system.

The `ibd` driver is dependent on GLD, a loadable kernel module that provides the `ibd` driver with the DLPI and STREAMS functionality required of a LAN driver. Except as noted in the `Application Programming Interface` section of this man page, see `gld(7D)` for more details on the primitives supported by the driver. The GLD module is located at `/kernel/misc/sparcv9/gld` on 64 bit systems and at `/kernel/misc/gld` on 32 bit systems.

The `ibd` driver expects certain configuration of the IBA fabric prior to operation (which also implies the SM must be active and managing the fabric). Specifically, the IBA multicast group representing the IPv4 limited broadcast address 255.255.255.255 (also defined as broadcast-GID in IETF documents) should be created prior to initializing the device. IBA properties (including mtu, qkey and sl) of this group is used by the driver to create any other IBA multicast group as instructed by higher level (IP) software. The driver probes for the existance of this broadcast-GID during `attach(9E)`.

Application Programming Interface (DLPI) The values returned by the driver in the DL_INFO_ACK primitive in response to your DL_INFO_REQ are:

- Maximum SDU is the MTU associated with the broadcast-GID group, less the 4 byte IPoIB header.
- Minimum SDU is 0.
- dlsap address length is 22.
- MAC type is DL_IB.
- The sap length value is -2, meaning the physical address component is followed immediately by a 2-byte sap component within the DLSAP address.

- Broadcast address value is the MAC address consisting of the 4 bytes of QPN 00:FF:FF:FF prepended to the IBA multicast address of the broadcast-GID.

  Due to the nature of link address definition for IPoIB, the DL_SET_PHYS_ADDR_REQ DLPI primitive is not supported.

  In the transmit case for streams that have been put in raw mode via the DLIOCRAW ioctl, the DLPI application must prepend the 20 byte IPoIB destination address to the data it wants to transmit over-the-wire. In the receive case, applications receive the IP/ARP datagram along with the IETF defined 4 byte header.

Warning This section describes warning messages that might be generated by the driver. Please note that while the format of these messages can be modified in future versions, the same general information is provided.

While joining IBA multicast groups corresponding to IP multicast groups as part of multicast promiscuous operations as required by IP multicast routers, or as part of running `snoop(1M)`, it is possible that joins to some multicast groups can fail due to inherent resource constraints in the IBA components. In such cases, warning message similar to the following appear in the system log, indicating the interface on which the failure occurred:

```
NOTICE: ibd0: Could not get list of IBA multicast groups
NOTICE: ibd0: IBA promiscuous mode missed multicast group
NOTICE: ibd0: IBA promiscuous mode missed new multicast gid
```

Also, if the IBA SM indicates that multicast trap support is suspended or unavailable, the system log contains a message similar to:

```
NOTICE: ibd0: IBA multicast support degraded due to
unavailability of multicast traps
```

And when the SM indicates trap support is restored:

```
NOTICE: ibd0: IBA multicast support restored due to
availability of multicast traps
```

Additionally, if the IBA link transitions to an unavailable state (that is, the IBA link state becomes `Down`, `Initialize` or `Armed`) and then becomes active again, the driver tries to rejoin previously joined groups if required. Failure to rejoin multicast groups triggers messages such as:

```
NOTICE: ibd0: Failure on port up to rejoin multicast gid
```

If the corresponding HCA port is in the unavailable state defined above when initializing an ibd interface using `ifconfig(1M)`, a message is emitted by the driver:

```
NOTICE: ibd0: Port is not active
```

Further, as described above, if the broadcast-GID is not found, or the associated MTU is higher than what the HCA port can support, the following messages are printed to the system log:

```
NOTICE: ibd0: IPoIB broadcast group absent
NOTICE: ibd0: IPoIB broadcast group MTU 4096 greater than port's
maximum MTU 2048
```

In all cases of these reported problems when running `ifconfig(1M)`, it should be checked that IBA cabling is intact, an SM is running on the fabric, and the broadcast-GID with appropriate properties has been created in the IBA partition.

The MTU of Reliable Connected mode can be larger than the MTU of Unreliable Datagram mode.

When Reliable Connected mode is enabled, `ibd` still uses Unreliable Datagram mode to transmit and receive multicast packets. If the payload size (excluding 4 byte IPoIB header) of a multicast packet is larger than the IP link MTU specified by the broadcast group, `ibd` drops it. A message appears in the system log when drops occur:

```
NOTICE: ibd0: Reliable Connected mode is on. Multicast packet
length (<packet length> > <IP_LINK_MTU>) is too long to send
```

If only one side has enabled Reliable Connected mode, communication falls back to datagram mode. The connected mode instance uses Path MTU discovery to automatically adjust the MTU of a unicast packet if an MTU difference exists. Before Path MTU discovery reduces the MTU for a specific destination, several packets whose size exceeds the MTU of Unreliable Datagram mode is dropped.

Configuration

The IPoIB service comes preconfigured on all HCA ports in the system. To turn the service off, or back on after turning it off, refer to documentation in `cfgadm_ib(1M)`.

Examples

EXAMPLE 1 Enabling Connected Mode

The following example `driver .conf` file enables Connected Mode for `ibd` instances `0` and `1`. Instances 2 and 3 use datagram mode.

```
# 1: unicast packets is sent over Reliable Connected Mode
# 0: unicast packets will be sent over Unreliable Datagram Mode
#
# Each element in the list below maps to the corresponding ibd
# instance; the first element is for ibd instance 0, the second
# element is for instance 1 and so on.
#
enable_rc=1,1,0,0
```

Files

`/dev/ibd*`	Special character device
`/kernel/drv/ib.conf`	Configuration file to start IPoIB service
`/kernel/drv/ibd.conf`	Configuration file for IPoIB driver
`/kernel/drv/sparcv9/ibd`	64–bit SPARC device driver
`/kernel/drv/amd64/ibd`	64–bit x86 device driver

`/kernel/drv/ibd`	32–bit x86 device driver

See Also `cfgadm(1M)`, `cfgadm_ib(1M)`, `ifconfig(1M)`, `syslogd(1M)`, `gld(7D)`, `ib(7D)`, `kstat(7D)`, `streamio(7I)`, `dlpi(7P)`, `attributes(5)`, `attach(9E)`

Notes IBD is a GLD-based driver and provides the statistics described by `gld(7D)`. Note that valid received packets not accepted by any stream (long) increases when IBD transmits broadcast IP packets. This happens because the infiniband hardware copies and loops back the transmitted broadcast packets to the source. These packets are discarded by GLD and are recorded as 'unknowns'.

Name ibdm – Solaris InfiniBand Device Manager

Description The Infiniband Device Manager (IBDM) is an IBTF-compliant kernel misc module. IBDM adheres to the InfiniBand Device Management class as described in *InfiniBand Architecture Specification, Volume 1: Release 1.1* and enumerates all the devices which are visible from a given host and maintains a data base of all IB devices visible to the host. IBDM provides interfaces to the IB nexus driver that enables the driver to retrieve information about IB devices on the fabric.

Files

`/kernel/misc/ibdm`	32-bit x86 ELF kernel module
`/kernel/misc/amd64/ibdm`	64-bit x86 ELF kernel module
`/kernel/misc/sparcv9/ibdm`	64-bit SPARC ELF kernel module

Attributes See `attributes(5)` for a description of the following attribute:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Contract Consolidation Private
Availability	SUNWhea

See Also `attributes(5)`, `ib(7D)`, `ibtl(7D)`, `ibcm(7D)`

InfiniBand Architecture Specification, Volume 1: Release 1.1

Diagnostics None.

Name ibdma – Solaris InfiniBand Device Manager Agent

Description The Infiniband Device Manager Agent (`ibdma`) is an IBTF-compliant kernel misc module.

`IBDMA` implements limited portions of the target (agent) side of the InfiniBand Device Management class as described in *InfiniBand Architecture Specification, Volume 1: Release 1.2.1*.

`IBDMA` responds to incoming Device Management Datagrams (MADS) by enumerating available target-side Infiniband services. Initiator systems can use this service to discover target-side resources such as the virtual I/O Controllers exported by `srpt(7D)`.

Files

`/kernel/misc/ibdma`	32-bit x86 ELF kernel module
`/kernel/misc/amd64/ibdma`	64-bit x86 ELF kernel module
`/kernel/misc/sparcv9/ibdma`	64-bit SPARC ELF kernel module

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWibdmar

See Also `attributes(5)`, `ib(7D)`, `ibdm(7D)`, `ibtl(7D)`, `srpt(7D)`

InfiniBand Architecture Specification, Volume 1: Release 1.2.1

Name ibmf – InfiniBand Management Transport Framework

Description The InfiniBand (IB) Management Transport Framework provides the mechanisms for IB management modules to communicate with other InfiniBand management modules such as the Subnet Administration process. It also provides helper functions such as Subnet Administration Access (SAA) for commonly performed operations.

Files

/kernel/misc/ibmf	32-bit ELF kernel misc module (x86 platform only).
/kernel/misc/amd64/ibmf	64-bit ELF kernel misc module (x86 platform only).
/kernel/misc/sparcv9/ibmf	64-bit ELF kernel misc module (SPARC platform only).

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	PCI-based systems
Interface stability	Consolidation Private
Availability	SUNWib

See Also `ibtl(7D)`

InfiniBand Architecture Specification, Version 1.1

www.infinibandta.org

Name ibtl – Solaris InfiniBand Transport Layer

Description InfiniBand (IB) is an I/O technology based on switched fabrics. The Solaris InfiniBand Transport Layer (`IBTL`) is a Solaris kernel misc module and adheres to the *IB Architecture Version 1.1* specification and provides a transport layer abstraction to IB client drivers.

`IBTL` implements the programming interfaces for the Solaris InfiniBand Transport Framework (IBTF), consisting of the IB Channel Interface (CI) and the IB Transport Interface (TI).

The CI consists of Host Channel Adapters (HCAs) and HCA drivers. A host is attached to the IB fabric through the CI layer. The Solaris InfiniBand CI is Sun's API rendering of the InfiniBand Architecture (IBTA) "verbs" specification.

The Solaris InfiniBand TI is the kernel service driver interface into the Solaris InfiniBand Transport Framework. It provides transport and communications setup programming interfaces for Unreliable Datagram (UD) and Reliable Connected (RC) transport types only.

Files

`/kernel/misc/ibtl`	32-bit x86 ELF kernel misc module
`/kernel/misc/amd64/ibtl`	64-bit x86 ELF kernel misc module
`/kernel/misc/sparcv9/ibtl`	64-bit SPARC ELF kernel module

Attributes See `attributes(5)` for a description of the following attribute:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Consolidation Private
Availability	SUNWhea, SUNWib

See Also `attributes(5)`, `ib(7D)`, `ibcm(7D)`, `ibdm(7D)`

InfiniBand Architecture Specification, Volume 1: Release 1.1

Name icmp6 – Internet Control Message Protocol for Internet Protocol Version 6

Synopsis

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip_icmp.h>
#include <netinet/icmp6.h>

s = socket(AF_INET6, SOCK_RAW, proto);

t = t_open("/dev/icmp6", O_RDWR);
```

Description The ICMP6 protocol is the error and control message protocol used with Version 6 of the Internet Protocol. It is used by the kernel to handle and report errors in protocol processing. It is also used for IPv6 neighbor and router discovery, and for multicast group membership queries and reports. It may also be accessed by programs using the socket interface or the Transport Level Interface (TLI) for network monitoring and diagnostic functions. When used with the socket interface, a "raw socket" type is used. The protocol number for ICMP6, used in the *proto* parameter to the socket call, can be obtained from `getprotobyname(3SOCKET)`. ICMP6 file descriptors and sockets are connectionless and are normally used with the `t_sndudata` / `t_rcvudata` and the `sendto()` / `recvfrom()` calls. They may also be used with the `sendmsg()`/`recvgmsg()` calls when sending or receiving ancillary data.

Outgoing packets automatically have an Internet Protocol Version 6 (IPv6) header and zero or more IPv6 extension headers prepended. These headers are prepended by the kernel. Unlike ICMP for IPv4, the `IP_HDRINCL` option is not supported for ICMP6, so ICMP6 applications neither build their own outbound IPv6 headers, nor do they receive the inbound IPv6 headers with received data. IPv6 extension headers and relevant fields of the IPv6 header may be set or received as ancillary data to a `sendmsg(3SOCKET)` or `recvmsg(3SOCKET)` system call. Each of these fields and extension headers may also be set on a per socket basis with the `setsockopt(3SOCKET)` system call. Such "sticky" options are used on all outgoing packets unless overridden by ancillary data. When any ancillary data is present with a `sendmsg(3SOCKET)` system call, all sticky options are ignored for that system call, but subsequently remain configured.

ICMP6 is a datagram protocol layered above IPv6. Received ICMP6 messages may be reflected back to users of higher-level protocols such as TCP or UDP as error returns from system calls. A copy of each ICMP6error message received by the system is provided to every holder of an open ICMP6 socket or TLI descriptor.

See Also `getprotobyname(3SOCKET)`, `recv(3SOCKET)`, `recvmsg(3SOCKET)`, `send(3SOCKET)`, `sendmsg(3SOCKET)`, `setsockopt(3SOCKET)`, `t_rcvudata(3NSL)`, `t_sndudata(3NSL)`, `inet6(7P)`, `ip6(7P)`, `routing(7P)`

Conta, A. and Deering, S., *RFC 2463, Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*, The Internet Society, December 1998.

Diagnostics A socket operation may fail with one of the following errors returned:

`EISCONN`	An attempt was made to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected.
`ENOTCONN`	An attempt was made to send a datagram, but no destination address is specified, and the socket has not been connected.
`ENOBUFS`	The system ran out of memory for an internal data structure.
`EADDRNOTAVAIL`	An attempt was made to create a socket with a network address for which no network interface exists.
`ENOMEM`	The system was unable to allocate memory for an internal data structure.
`ENOPROTOOPT`	An attempt was made to set an IPv4 socket option on an IPv6 socket.
`EINVAL`	An attempt was made to set an invalid or malformed socket option.
`EAFNOSUPPORT`	An attempt was made to bind or connect to an IPv4 or mapped address, or to specify an IPv4 or mapped address as the next hop.

Name icmp, ICMP – Internet Control Message Protocol

Synopsis

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip_icmp.h>
s = socket(AF_INET, SOCK_RAW, proto);
t = t_open("/dev/icmp", O_RDWR);
```

Description ICMP is the error and control message protocol used by the Internet protocol family. It is used by the kernel to handle and report errors in protocol processing. It may also be accessed by programs using the socket interface or the Transport Level Interface (TLI) for network monitoring and diagnostic functions. When used with the socket interface, a "raw socket" type is used. The protocol number for ICMP, used in the *proto* parameter to the socket call, can be obtained from `getprotobyname(3SOCKET)`. ICMP file descriptors and sockets are connectionless, and are normally used with the `t_sndudata` / `t_rcvudata` and the `sendto()` / `recvfrom()` calls.

Outgoing packets automatically have an Internet Protocol (IP) header prepended to them. Incoming packets are provided to the user with the IP header and options intact.

ICMP is an datagram protocol layered above IP. It is used internally by the protcol code for various purposes including routing, fault isolation, and congestion control. Receipt of an ICMP "redirect" message will add a new entry in the routing table, or modify an existing one. ICMP messages are routinely sent by the protocol code. Received ICMP messages may be reflected back to users of higher-level protocols such as TCP or UDP as error returns from system calls. A copy of all ICMP message received by the system is provided to every holder of an open ICMP socket or TLI descriptor.

See Also `getprotobyname(3SOCKET)`, `recv(3SOCKET)`, `send(3SOCKET)`, `t_rcvudata(3NSL)`, `t_sndudata(3NSL)`, `inet(7P)`, `ip(7P)`, `routing(7P)`

Postel, Jon, *Internet Control Message Protocol — DARPA Internet Program Protocol Specification*, RFC 792, Network Information Center, SRI International, Menlo Park, Calif., September 1981.

Diagnostics A socket operation may fail with one of the following errors returned:

`EISCONN`	An attempt was made to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected.
`ENOTCONN`	An attempt was made to send a datagram, but no destination address is specified, and the socket has not been connected.
`ENOBUFS`	The system ran out of memory for an internal data structure.
`EADDRNOTAVAIL`	An attempt was made to create a socket with a network address for which no network interface exists.

Notes Replies to ICMP "echo" messages which are source routed are not sent back using inverted source routes, but rather go back through the normal routing mechanisms.

Name idn – inter-domain network device driver

Synopsis `/dev/idn`

Description The `idn` driver is a multi-thread, loadable, clonable, STREAMS-based pseudo driver that supports the connectionless Data Link Provider Interface `dlpi(7P)` over the Sun Enterprise 10000 Gigplane-XB Interconnect. This connection is permitted only between domains within the same Sun Enterprise 10000 server.

The `idn` driver supports 1 to 32 logical network interfaces that can be connected to domains linked to the local domain through the `domain_link`(1M) command. (See `domain_link`(1M) in the for more information.) The `idn` driver works in conjunction with the System Service Processor (SSP) to perform domain linking/unlinking and automated linking upon host bootup.

The `/dev/idn` device is used to access all IDN services provided by the system.

IDN and DLPI The `idn` driver is a style-2 Data Link Service provider. All `M_PROTO` and `M_PCPROTO`–type messages are interpreted as DLPI primitives. For the `idn` driver to associate the opened stream with a particular device (ppa), you must send an explicit `DL_ATTACH_REQ` message. The ppa ID is interpreted as an unsigned long and indicates the corresponding device instance (unit) number. The `DL_ERROR_ACK` error is returned by the driver if the ppa field value does not correspond to a valid device-instance number for the system. The device is initialized on first attach and de-initialized (stopped) on the last detach.

- The maximum SDU is configurable by using the `idn.conf` file and has a range of 512 bytes to 512 Kbytes. The default value is 16384 bytes.
- The minimum SDU is 0.
- The Service Access Pointer (SAP) address length is 8.
- The MAC type is `DL_ETHER`.
- The SAP length value is *-2*, meaning the physical address component is followed immediately by a 2-byte SAP component within the `DLSAP` address.
- The service mode is `DL_CLDLS`.
- Optional quality of service (QOS) is not presently supported; accordingly, the QOS fields are 0.
- The provider style is `DL_STYLE2`.
- The version is `DL_VERSION_2`.
- The broadcast address value is Ethernet/IEEE broadcast address (0xFFFFFF). The `idn` driver supports broadcast by issuing messages to each target individually. The `idn` driver is inherently a point-to-point network between domains. When the `idn` driver is in the `DL_ATTACHED` state, the user must send a `DL_BIND_REQ` request to associate a particular SAP with the stream. The `idn` driver interprets the SAP field within the `DL_BIND_REQ` message

as an Ethernet type and valid values for the SAP field are in the range of *0* to *0xFFFF*. Only one Ethernet type can be bound to the stream at any time.

If a SAP with a value of 0 is selected, the receiver will be in 802.3 mode. All frames received from the media having a type field in the range of 0 to 1500 are assumed to be 802.3 frames and are routed up all open streams which are bound to SAP value 0. If more than one stream is in 802.3 mode, then the frame will be duplicated and routed up as multiple stream `DL_UNITDATA_IND` messages.

In transmission, the driver checks the SAP field of the `DL_BIND_REQ` to determine if the SAP value is 0, and if the destination type field is in the range of *0* to *1500*. If either is true, the driver computes the length of the message, (excluding the initial message block `M_PROTO` `mblk`) of all subsequent `DL_UNITDATA_REQ` messages and transmits 802.3 frames that have this value in the MAC frame header length field.

The driver also supports raw `M_DATA` mode. When the user sends a `DLIOCRAW` ioctl, the particular stream is put in raw mode. A complete frame and a proper ether header is expected as part of the data.

The DLSAP address format consists of the 6-byte, physical address component (Ethernet) followed immediately by the 2-byte SAP component (type), producing an 8-byte DLSAP address. Applications should *not* hardcode to this particular implementation-specific DLSAP address format, but instead should use information returned in the `DL_INFO_ACK` primitive to compose and decompose DLSAP addresses. The SAP length, full DLSAP length, and SAP physical ordering are included within the `DL_INFO_ACK` primitive. The physical address length can be computed by subtracting the SAP length from the full DLSAP address length or by issuing the `DL_PHYS_ADDR_REQ` message to obtain the current physical address associated with the stream.

When the `idn` driver is in the `DL_BOUND` state, you can transmit frames on the IDN by sending `DL_UNITDATA_REQ` messages to the driver. The driver then routes received IDN frames up the open and bound streams having a SAP which matches the Ethernet type as `DL_UNITDATA_IND` messages. If necessary, received IDN frames are duplicated and routed up multiple open streams. The DLSAP address contained within the `DL_UNITDATA_REQ` and `DL_UNITDATA_IND` messages consists of both the SAP (type) and physical (Ethernet) components.

IDN Primitives

In addition to the mandatory connectionless DLPI message set, the `idn` driver supports the following primitives:

The `DL_ENABMULTI_REQ` and `DL_DISABMULTI_REQ` primitives which enable or disable, respectively, the reception of individual multicast group addresses. A set of multicast addresses may be iteratively created and modified on a per-stream basis using these primitives. These primitives are accepted by the driver in any state following the `DL_ATTACHED` state.

The DL_PROMISCON_REQ and DL_PROMISCOFF_REQ primitives, which with the DL_PROMISC_PHYS flag set in the dl_level field, enable or disable, respectively, the reception of all promiscuous frames on the media, including frames generated by the local domain. When used with the DL_PROMISC_SAP flag set in the dl_level field, these primitives enable or disable, respectively, the reception of all SAP (Ethernet type) values. When used with the DL_PROMISC_MULTI flag set in the dl_level field, these primitives enable or disable, respectively, the reception of all multicast group addresses. The effect of each is always on a per-stream basis and independent of the other SAP and physical level configurations on this stream or other streams.

The DL_PHYS_ADDR_REQ primitive which returns the 6-octet, Ethernet address associated with (or attached to) the stream in the DL_PHYS_ADDR_ACK primitive. This primitive is valid only in states following a successful DL_ATTACH_REQ request.

Because the driver maintains domain address information in the address to direct packets to the correct destination, the DL_SET_PHYS_ADDR_REQ primitive is not allowed.

Files The following files are supported:

/dev/idn
IDN special character device

/platform/SUNW,Ultra-Enterprise-10000/kernel/drv/idn.conf
System-wide and per-interface default device driver properties

See Also netstat(1M), ndd(1M), dlpi(7P)

domain_link(1M) in the .

Notes The idn driver supports a set of properties that can be set by using the driver.conf file for the IDN. See the for more information about the properties in the driver.conf(4), (idn.conf, for IDNs).

Name iec61883 – IEC 61883 interfaces

Synopsis

```
#include <sys/av/iec61883.h>
```

Description The set of interfaces described in this man page can be used to control and exchange data with consumer audio/video devices using protocols specified in*IEC 61883 Consumer Electronic Audio/Video Equipment - Digital Interface*, including Common Isochronous Packet (CIP), Connection Management Procedures (CMP) and Function Control Protocol (FCP).

An `iec61883` compliant driver exports two device nodes for isochronous and for asynchronous transactions. See the `FILES` section of this man page for the namespace definition.

Isochronous Transfers Two methods are provided to receive/transmit isochronous data: using `mmap(2)` in combination with `ioctl(2)`, and `read(2)` or `write(2)`.

Mmap/Ioctl This method provides better performance and finer-grained control than read/write, and is a method of choice for most applications. The data buffer is mapped into a user process address space, which means no data copying between the kernel and an application is necessary. Synchronization between user processes and the driver is performed using `ioctl(2)` commands.

An application allocates resources for isochronous transfer using `IEC61883_ISOCH_INIT`. Then the data buffer can be mapped into the process space using `mmap(2)`.

A circular data buffer consists of one or more equal size frame buffers (further referred to as frames, unless to avoid ambiguity with AV frames). Frames are numbered starting with zero and are always transferred sequentially. Frames consist equal sized packets. Each packet contains a CIP header and one or more data blocks.

A driver and an application act as a producer and a consumer: producer supplies *full* frames (filled with data) to the consumer, and the producer is not allowed to access those frames until the consumer claims them *empty*.

A transfer can be initiated and suspended with `IEC61883_START` and `IEC61883_STOP` commands respectively. `IEC61883_RECV` or `IEC61883_XMIT` is used for producer-consumer synchronization.

Read/Write Using this method, an application calls `read(2)` or `write(2)` to receive or transmit a specified amount of data. Bus-specific overhead, such as isochronous packet headers, is handled by the driver and is not exposed to applications. Data returned by `read(2)` contains CIP headers and data blocks. Empty packets are not returned by `read(2)`. `write(2)` data should meet the same requirements.

If one or more channels have been allocated since `open(2)` (see `IEC61883_ISOCH_INIT`), the data is received/transmitted using channel that was created the last.

If no channels were allocated, the driver uses the broadcast channel by default and allocates the default-size data buffer. During transmit, the first packet's CIP header is used to auto-detect the data format. If it is one of the formats supported by the driver, it is properly transmitted (with inserted empty packets and timestamps).

For both methods, if during transmit the driver runs out of data, it transmits empty packets containing only a CIP header of the next to be transmitted packet, as defined in *IEC 61883-1*.

Connection Management Procedures

Applications wishing to follow Connection Management Procedures (CMP) in combination with isochronous transfers should use the `ioctl(2)` `IEC61883_PLUG_INIT`, `IEC61883_PLUG_FINI`, `IEC61883_PLUG_REG_READ` and `IEC61883_PLUG_REG_CAS` commands.

Asynchronous Transactions

`read(2)`, `write(2)`, `ioctl(2)`, and `poll(2)` can be used with asynchronous nodes. Asynchronous data exchange between a driver and an application utilizes a common data structure called asynchronous request (ARQ):

```
typedef struct iec61883_arq {
        int        arq_type;
        int        arq_len;
        union {
                uint32_t   quadlet;
                uint64_t   octlet;
                uint8_t    buf[8];
        } arq_data;
} iec61883_arq_t;
```

`arq_type` contains ARQ type:

`IEC61883_ARQ_FCP_CMD`
`IEC61883_ARQ_FCP_RESP`
: FCP command and response frame respectively. Outgoing frames are sent using `write(2)`, incoming frames are received with `read(2)`.

  See *IEC 61883-1* for the FCP frame structure definition.

`IEC61883_ARQ_BUS_RESET`
: Returned by the driver when a bus reset occurs. There is no data associated with this request type, and `arq_len` is set to 0.

If `arq_len` is 4 or 8, then data should be supplied in `arq_data.quadlet` or `arq_data.octlet` respectively, otherwise up to 8 bytes can be put in `arq_data.buf`, with the rest of the data following immediately after.

write(2)

For a request to be sent to a target, an `iec61883_arq_t` structure along with associated data is passed to the driver using `write(2)`. `write()` blocks until the request is completed.

read(2) A driver collects incoming ARQs in the internal buffer. Buffer size can be changed using the `ioctl(2)` command `IEC61883_FCP_SET_IBUF_SIZE`.

Reading an ARQ takes one or two steps depending on data length. An application first reads `sizeof (iec61883_arq_t)` bytes: if `arq_len` is less than or equal 4, which is usually the case, no additional step is needed. Otherwise, the remaining `arq_len - 4` bytes should be read and concatenated.

`read(2)` blocks until the specified amount of data is available, unless `O_NONBLOCK` or `O_NDELAY` flag was set during `open(2)`, in which case`read(2)` returns immediately.

poll(2) Applications can `poll(2)` asynchronous nodes on the `POLLIN` event.

Bus Reset In case of a bus reset, the driver notifies an application by generating an `ARQ` of type `IEC61883_ARQ_BUS_RESET`.

If there were established isochronous connections before bus reset, the driver attempts to restore all connections as described in *IEC 61883* and resume any active transfers that were in progress.

ioctls The following commands only apply to isochronous nodes:

`IEC61883_ISOCH_INIT`
This command allocates a data buffer and isochronous resources (if necessary) for the isochronous transfer. The argument is a pointer to the structure:

```
typedef struct iec61883_isoch_init {
      int ii_version;       /* interface version */
      int   ii_pkt_size;    /* packet size */
      int   ii_frame_size;  /* packets/frame */
      int   ii_frame_cnt;   /* # of frames */
      int   ii_direction;   /* xfer direction */
      int   ii_bus_speed;   /* bus speed */
      uint64_t ii_channel;  /* channel mask */
      int   ii_dbs;         /* DBS */
      int   ii_fn;          /* FN */
      int   ii_rate_n;      /* rate numerator */
      int   ii_rate_d;      /* rate denominator */
      int   ii_ts_mode;     /* timestamp mode */
      int   ii_flags;       /* flags */
      int   ii_handle;      /* isoch handle */
      int   ii_frame_rcnt;  /* # of frames */
      off_t   *ii_mmap_off  /* mmap offset */
      int   ii_rchannel;    /* channel */
      int   ii_error;       /* error code */
} iec61883_isoch_init_t;
```

`ii_version` should be set to `IEC61883_V1_0`.

The driver attempts to allocate a data buffer consisting of `ii_frame_cnt` frames, with `ii_frame_size` packets in each frame. Packet size in bytes is specified by `ii_pkt_size` specifies and should be a multiple of 512 and compatible with `ii_bus_speed`.

`ii_direction` can take one of the following values:

`IEC61883_DIR_RECV`
Receiving isochronous data.

`IEC61883_DIR_XMIT`
Transmitting isochronous data.

`ii_bus_speed` chooses bus speed to be used and can be either `IEC61883_S100`, `IEC61883_S200` or `IEC61883_S400`.

`ii_channel` is a mask that specifies an isochronous channel number to be used, with the *N*th bit representing channel *N*. When transmitting data, several bits can be set at a time, in which case the driver chooses one, for example, `0x3FF` means a range from 0 to 9. In case of receive, only one bit can be set.

`ii_dbs` specifies data block size in quadlets, for example, DBS value for `SD-DVCR` is `0x78`. Refer to *IEC 61883* for more details on DBS.

ii_fn specifies fraction number, which defines the number of blocks in which a source packet is divided. Allowed values are from 0 to 3. Refer to IEC 61883 for more details on FN.

Data rate expected by the AV device can be lower than the bus speed, in which case the driver has to periodically insert empty packets into the data stream to avoid device buffer overflows. This rate is specified with a fraction N/D, set by `ii_rate_n` and `ii_rate_d` respectively. Any integer numbers can be used, or the following predefined constants:

`IEC61883_RATE_N_DV_NTSC IEC61883_RATE_D_DV_NTSC`
Data rate expected by `DV-NTSC` devices.

`IEC61883_RATE_N_DV_PAL IEC61883_RATE_D_DV_PAL`
Data rate expected by `DV-PAL` devices.

During data transmission, a timestamp based on the current value of the cycle timer is usually required. `ii_ts_mode` defines timestamp mode to be used:

`IEC61883_TS_SYT`
Driver puts a timestamp in the SYT field of the first CIP header of each frame.

`IEC61883_TS_NONE`
No timestamps.

`ii_dbs`, `ii_fn`, `ii_rate_n`, `ii_rate_d` and `ii_ts_mode` are only required for transmission. In other case these should be set to 0.

`ii_flags` should be set to 0.

If command succeeds, `ii_handle` contains a handle that should be used with other isochronous commands. `ii_frame_rcnt` contains the number of allocated frames (can be less than `ii_frame_cnt`). `ii_mmap_off` contains an offset to be used in `mmap(2)`, for example, to map an entire data receive buffer:

```
pa = mmap(NULL, init.ii_pkt_size *
      init.ii_frame_size * init.ii_frame_rcnt,
      PROT_READ, MAP_PRIVATE, fd, init.ii_mmap_off);
```

`ii_rchannel` contains channel number.

In case of command success, `ii_error` is set to 0; otherwise one of the following values can be returned:

`IEC61883_ERR_NOMEM`
Not enough memory for the data buffer.

`IEC61883_ERR_NOCHANNEL`
Cannot allocate isochronous channel.

`IEC61883_ERR_PKT_SIZE`
Packet size is not allowed at this bus speed.

`IEC61883_ERR_VERSION`
Interface version is not supported.

`IEC61883_ERR_INVAL`
One or more the parameters are invalid

`IEC61883_ERR_OTHER`
Unspecified error type.

`IEC61883_ISOCH_FINI`
Argument is a handle returned by `IEC61883_ISOCH_INIT`. This command frees any resources associated with this handle. There must be no active transfers and the data buffer must be unmapped; otherwise the command fails.

`IEC61883_START`
This command starts an isochronous transfer. The argument is a handle returned by `IEC61883_ISOCH_INIT`.

`IEC61883_STOP`
This command stops an isochronous transfer. The argument is a handle returned by `IEC61883_ISOCH_INIT`.

`IEC61883_RECV`
This command is used to receive full frames and return empty frames to the driver. The argument is a pointer to the structure:

```
typedef struct iec61883_recv {
        int rx_handle;     /* isoch handle */
        int rx_flags;      /* flags */
iec61883_xfer_t rx_xfer;   /* xfer params */
} iec61883_recv_t;

typedef struct iec61883_xfer {
        int   xf_empty_idx; /* first empty frame */
        int xf_empty_cnt;   /* empty frame count */
        int   xf_full_idx;  /* first full frame */
        int   xf_full_cnt;  /* full frame count */
        int   xf_error;     /* error */
} iec61883_xfer_t;
```

`rx_flags` should be set to 0.

An application sets `xf_empty_idx` and `xf_empty_cnt` to indicate frames it no longer needs. E.g. if a buffer consists of 6 frames, `xf_empty_idx` is 4, `xf_empty_cnt` is 3 - means that frames 4, 5 and 0 can now be reused by the driver. If there are no empty frames, for example, the first time this command is called, `xf_empty_cnt` should be set to 0.

When the command returns, `xf_full_idx` and `xf_full_cnt` specifies the frames that are full. `xf_error` is always 0.

In general, AV frame boundaries are not aligned with the frame buffer boundaries, because the first received packet might not be the first packet of an AV frame, and, in contrast with the read/write method, the driver does not remove empty CIP packets.

Applications should detect empty packets by comparing adjacent packets' continuity counters (DBC field of the CIP header).

`IEC61883_XMIT`

This command is used to transmit full frames and get more empty frames from the driver. The argument is a pointer to the structure:

```
typedef struct iec61883_xmit {
        int   tx_handle;         /* isoch handle */
        int   tx_flags;          /* flags */
        iec61883_xfer_t tx_xfer; /* xfer params */
        int   tx_miss_cnt;       /* missed cycles */
 } iec61883_xmit_t;
```

`tx_flags` should be set to zero.

The application sets `xf_full_idx` and `xf_full_cnt` to specify frames it wishes to transmit. If there are no frames to transmit (e.g. the first time this command is called), `xf_full_cnt` should be set to 0.

When the command returns, `xf_empty_idx` and `xf_empty_cnt` specifies empty frames which can be to transmit more data. `xf_error` is always 0.

tx_miss_cnt contains the number of isochronous cycles missed since last transfer due to data buffer under run. This can happen when an application does not supply data fast enough.

For the purposes of time stamping, the driver considers the first packet in a frame buffer to be the first packet of an AV frame.

IEC61883_PLUG_INIT
This command returns a handle for the specified plug. The argument is a pointer to the structure:

```
typedef struct iec61883_plug_init {
        int   pi_ver;     /* interface version */
        int   pi_loc;     /* plug location */
        int   pi_type;    /* plug type */
        int   pi_num;     /* plug number */
        int   pi_flags;   /* flags */
        int   pi_handle;  /* plug handle */
        int   pi_rnum;    /* plug number */
 } iec61883_plug_init_t;
```

pi_ver should be set to IEC61883_V1_0.

pi_loc specifies plug location:

IEC61883_LOC_LOCAL
On the local unit (local plug). A plug control register (PCR) is allocated. Command fails if the plug already exists

IEC61883_LOC_REMOTE
On the remote unit (remote plug). The plug should exist on the remote unit, otherwise the command fails.

pi_type specifies isochronous plug type:

IEC61883_PLUG_IN IEC61883_PLUG_OUT
Input or output plugs.

IEC61883_PLUG_MASTER_IN IEC61883_PLUG_MASTER_OUT
Master input or master output plug. These plugs always exist on the local unit.

pi_num specifies plug number. This should be 0 for master plugs, and from 0 to 31 for input/output plugs. Alternatively, a special value IEC61883_PLUG_ANY can be used to let the driver choose a free plug number, create the plug and return the number in pi_rnum.

pi_flags should be set to 0.

If the command succeeds, pi_handle contains a handle that should be used with other plug commands.

`IEC61883_PLUG_FINI`
Argument is a handle returned by `IEC61883_PLUG_INIT`. This command frees any resources associated with this handle, including the PCR.

`IEC61883_PLUG_REG_READ`
Read plug register value. The argument is a pointer to the structure:

```
typedef struct iec61883_plug_reg_val {
        int         pr_handle; /* plug handle */
        uint32_t     pr_val;    /* register value */
} iec61883_plug_reg_val_t;
```

`pr_handle` is a handle returned by `IEC61883_PLUG_INIT`. Register value is returned in `pr_val`.

`IEC61883_PLUG_REG_CAS`
Atomically compare and swap plug register value. The argument is a pointer to the structure:

```
typedef struct iec61883_plug_reg_lock {
        int        pl_handle; /* plug handle */
        uint32_t   pl_arg;    /* compare arg */
        uint32_t   pl_data;   /* write value */
        UINT32_t   pl_old;    /* original value */
} iec61883_plug_reg_lock_t;
```

pr_handle is a handle returned by IEC61883_PLUG_INIT.

Original register value is compared with pl_arg and if they are equal, register value is replaced with pl_data. In any case, the original value is stored in pl_old.

The following commands only apply to asynchronous nodes:

`IEC61883_ARQ_GET_IBUF_SIZE`
This command returns current incoming ARQ buffer size. The argument is a pointer to `int`.

`IEC61883_ARQ_SET_IBUF_SIZE`
This command changes incoming ARQ buffer size. The argument is the new buffer size in bytes.

Files

`/dev/av/N/async`	Device node for asynchronous data
`/dev/av/N/isoch`	Device has been disconnected

Errors

`EIO`	Bus operation failed.
	DMA failure.
`EFAULT`	`ioctl(2)` argument points to an illegal address.
`EINVAL`	Invalid argument or argument combination.

ENODEV Device has been disconnected.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	All
Stability level	Committed

See Also ioctl(2), mmap(2), open(2), poll(2), read(2), write(2), attributes(5), av1394(7D)

IEC 61883 Consumer audio/video equipment - Digital interface

IEEE Std 1394-1995 Standard for a High Performance Serial Bus

Name ieee1394, firewire, 1394 – Solaris IEEE-1394 Architecture

Description IEEE-1394 provides a means for interconnecting devices in computer and home entertainment systems. (The IEEE-1394 architecture is also known as Firewire, an Apple Computer trademark, and i.Link, a Sony trademark). The most common IEEE-1394 devices are digital camcorders, mass-storage devices and cameras (including webcam-type devices). For more information on USB, refer to the 1394 Trade Association website at http://*www.1394ta.org*.

The Solaris IEEE-1394 architecture supports up to 63 hot-pluggable IEEE-1394 devices per IEEE-1394 bus. The maximum data transfer rate is 400 Mbits, depending on the capabilities of the attached device.

The Solaris IEEE-1394 architecture supports devices implementing a number of different specifications. The basic behavior of the IEEE-1394 bus is described in the *IEEE 1394-1995* and *IEEE 1394a-2000* specifications.

IEEE-1394 host controllers implementing the 1394 Open Host Controller Interface specification are supported. Camcorders implementing the *IEC 61883* and 1394 Trade Association AV/C specifications are supported. Mass-storage devices implementing the *ANSI SBP-2* specification are supported. Digital cameras implementing the 1394 Trade Association 1394-based Digital Camera (IIDC) specification are supported.

Files Listed below are drivers and modules which either utilize or are utilized by the Solaris IEEE-1394 architecture. Drivers in `/kernel/drv` are 32 bit drivers (only). Drivers in `/kernel/drv/sparcv9` or `/kernel/drv/amd64` are 64 bit drivers.

SUPPORT MODULE(S)	FUNCTION
/kernel/misc/[sparcv9|amd64/]s1394	IEEE-1394 framework
/kernel/misc/[sparcv9|amd64/]sbp2	Serial Bus Protocol-2 (SBP-2)

TARGET DRIVER	DEVICE CLASS
/kernel/drv/[sparcv9|amd64/]s1394	IEEE-1394 framework
/kernel/drv/[sparcv9|amd64/]scsa1394	mass storage class
/kernel/drv/[sparcv9|amd64/]av1394	camcorder (AV/C) class
/kernel/drv/[sparcv9|amd64/]dcam1394	digital camera (IIDC) class

HOST CONTROLLER INTERFACE DRIVER(S)	DEVICE
/kernel/drv/[sparcv9|amd64/]hci1394	Open HCI

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	PCI-based systems
Availability	SUNW1394, SUNW1394h, SUNWav1394, SUNWscsa1394, SUNWsbp2, SUNWfwdc, SUNWfwdcu

See Also attributes(5), av1394(7D), dcam1394(7D), hci1394(7D), scsa1394(7D), iec61883(7I)

http://www.sun.com

IEEE 1394a Specification – 1394 Trade Association, 2000

IEEE 1394 Specification – 1394 Trade Association, 1995

Notes Booting from IEEE-1394 mass-storage devices is not supported, but may be possible if supported by the BIOS of the computer system.

Name ifb – 24-bit PCI color frame buffer and graphics accelerator driver

Description The `ifb` driver is the device driver for the Sun Expert3D graphics accelerators. The Expert3D is a high resolution, high performance PCI graphics framebuffer providing hardware texture mapping. The Expert3D also supports 1920x1200 double buffered, z-buffered display and 1280 x 1024 stereo.

The `ifbdaemon` process loads the `ifb` microcode at system startup time and during the resume sequence of a suspend-resume cycle.

Files `/dev/fbs/ifb`*n* Device special file

`/usr/lib/ifb.ucode` ifb microcode

`/usr/sbin/ifbdaemon` ifb microcode loader

See Also `SUNWifb_config(1M)`

Name ifp – ISP2100 Family Fibre Channel Host Bus Adapter Driver

Synopsis

```
PCI    SUNW,ifp@pci-slot
```

Description The `ifp` Host Bus Adapter is a SCSA compliant nexus driver for the Qlogic ISP2100/ISP2100A chips. These chips support Fibre Channel Protocol for SCSI on Private Fibre Channel Arbitrated loops.

The `ifp` driver interfaces with SCSI disk target driver, `ssd(7D)`, and the SCSI-3 Enclosure Services driver, `ssd(7D)`. Only SCSI devices of type `disk` and `ses` are supported at present time.

The `ifp` driver supports the standard functions provided by the SCSA interface. It supports auto request sense (cannot be turned off) and tagged queueing by default. The driver requires that all devices have unique hard addresses defined by switch settings in hardware. Devices with conflicting hard addresses will not be accessible.

Files

`/kernel/drv/ifp`	ELF Kernel Module
`/kernel/drv/sparcv9/ifp`	ELF Kernel Module (64–bit version)
`/kernel/drv/ifp.conf`	Driver configuration file

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SPARC

See Also `luxadm(1M)`,`prtconf(1M)`,`driver.conf(4)`,`attributes(5)`,`ses(7D)`,`ssd(7D)`

Writing Device Drivers,

ANSI X3.272–1996, Fibre Channel Arbitrated Loop (FC-AL),

ANSI X3.269-1996, Fibre Channel Protocol for SCSI (FCP),

ANSI X3.270-1996, SCSI-3 Architecture Model (SAM),

Fibre Channel Private Loop SCSI Direct Attach (FC-PLDA),

ISP2100 Firmware Interface Specification, QLogic Corporation

Diagnostics The messages described below are some that may appear on the system console, as well as being logged.

This first set of messages may be displayed while the ifp driver is initially trying to attach. All of these messages mean that the ifp driver was unable to attach. These messages are preceded by "`ifp`<*number*>", where "<*number*>" is the instance number of the ISP2100 Host Bus Adapter.

Device is using a hilevel intr, unused	The device was configured with an interrupt level that cannot be used with this ifp driver. Check the device.
Failed to alloc soft state	Driver was unable to allocate space for the internal state structure. Driver did not attach to device; SCSI devices will be inaccessible.
Bad soft state	Driver requested an invalid internal state structure. Driver did not attach to device; SCSI devices will be inaccessible.
Unable to map pci config registers Unable to map biu registers	Driver was unable to map device registers; check for bad hardware. Driver did not attach to device; SCSI devices will be inaccessible.
Cannot alloc tran	Driver was unable to obtain a transport handle to be able to communicate with SCSA framework. Driver did not attach to device; SCSI devices will be inaccessible.
ddi_create_minor_node failed	Driver was unable to create `devctl` minor node that is used by `luxadm(1M)` for administering the loop. Driver did not attach to device; SCSI devices will be inaccessible.
Cannot alloc dma handle	Driver was unable allocate a `dma` handle for communicating with the Host Bus Adapter. Driver did not attach to device; SCSI devices will be inaccessible.
Cannot alloc cmd area	Driver was unable allocate dma memory for request and response queues. Driver did not attach to device; SCSI devices will be inaccessible.
Cannot bind cmd area	Driver was unable to bind `dma` handle to the `cmd` area. Driver did not attach to device; SCSI devices will be inaccessible.
Cannot alloc fcal handle	Driver was unable allocate a `dma` handle for retrieving loop map from the Host Bus Adapter. Driver did not attach to device; SCSI devices will be inaccessible.
Cannot bind portdb	Driver was unable to bind `fcal` port handle to the memory used for obtaining port database. Driver did not attach to device; SCSI devices will be inaccessible.

scsi_hba_attach failed	Driver was unable to attach to the SCSA framework. Driver did not attach to device; SCSI devices will be inaccessible.
Unable to create hotplug thread	Driver was not able to create the kernel thread used for hotplug support. Driver did not attach to device; SCSI devices will be inaccessible.
Cannot add intr	Driver was not able to add the interrupt routine to the kernel. Driver did not attach to device; SCSI devices will be inaccessible.
Unable to attach	Driver was unable to attach to the hardware for some reason that may be printed. Driver did not attach to device; SCSI devices will be inaccessible.

The following set of messages may be display at any time. They will be printed with the full device pathname followed by the shorter form described above.

Firmware checksum incorrect	Firmware has an invalid checksum and will not be downloaded.
Chip reset timeout	ISP chip failed to reset in the time allocated; may be bad hardware.
Stop firmware failed	Stopping the firmware failed; may be bad hardware.
Load ram failed	Unable to download new firmware into the ISP chip.
DMA setup failed	The DMA setup failed in the host adapter driver on a `scsi_pkt`. This will return `TRAN_BADPKT` to a SCSA target driver.
Bad request pkt type Bad request pkt Bad request pkt hdr Bad req pkt order	The ISP Firmware rejected the packet as being set up incorrectly. This will cause the ifp driver to call the target completion routine with the reason of `CMD_TRAN_ERR` set in the `scsi_pkt`. Check the target driver for correctly setting up the packet.
Firmware error	The ISP chip encountered a firmware error of some kind. This error will cause the ifp driver to do error recovery by resetting the chip.
DMA Failure (event)	The ISP chip encountered a DMA error while reading from the request queue (event is 8003) or writing to the

	response queue (event is 8004). This error will cause the ifp driver to do error recovery by resetting the chip.
Fatal error, resetting interface	This is an indication that the `ifp` driver is doing error recovery. This will cause all outstanding commands that have been transported to the `ifp` driver to be completed via the `scsi_pkt` completion routine in the target driver with reason of `CMD_RESET` and status of `STAT_BUS_RESET` set in the `scsi_pkt`.
target t, duplicate port wwns	The driver detected target *t* to be having the same port WWN as a different target; this is not supposed to happen. Target *t* will become inaccessible.
target t, duplicate switch settings	The driver detected devices with the same switch setting *t*. All such devices will become inaccessible.
WWN changed on target t	The World Wide Name (WWN) has changed on the device with switch setting *t*.
target t, unknown device type dt	The driver does not know the device type *dt* reported by the device with switch setting *t*.

Name if_tcp, if – general properties of Internet Protocol network interfaces

Description A network interface is a device for sending and receiving packets on a network. It is usually a hardware device, although it can be implemented in software. Network interfaces used by the Internet Protocol (IPv4 or IPv6) must be STREAMS devices conforming to the Data Link Provider Interface (DLPI). See `dlpi(7P)`.

Application Programming Interface An interface becomes available to IP when it is opened and the IP module is pushed onto the stream with the `I_PUSH` `ioctl(2)` command. (See `streamio(7I)`). The `SIOCSLIFNAME` `ioctl(2)` is issued to specify the name of the interface and to indicate whether it is IPv4 or IPv6. This may be initiated by the kernel at boot time or by a user program after the system is running. Each interface must be assigned an IP address with the `SIOCSLIFADDR` `ioctl()` before it can be used. On interfaces where the network-to-link layer address mapping is static, only the network number is taken from the `ioctl()` request; the remainder is found in a hardware specific manner. On interfaces which provide dynamic network-to-link layer address mapping facilities (for example, Ethernets using `arp(7P)`), the entire address specified in the `ioctl()` is used. A routing table entry for destinations on the network of the interface is installed automatically when an interface's address is set.

You cannot create IPMP IP interfaces using the procedure described above. Instead, use `ifconfig(1M)`.

ioctls The following `ioctl()` calls may be used to manipulate IP network interfaces. Unless specified otherwise, the request takes an `lifreq` structure as its parameter. This structure has the form:

```
struct lifreq {
#define LIFNAMSIZ    32
   char    lifr_name[LIFNAMSIZ];       /* if name, e.g. "le1" */
   union {
      int    lifru_addrlen;            /* for subnet/token etc */
      uint_t lifru_ppa;                /* SIOCSLIFNAME */
   } lifr_lifru1;
   union {
      struct sockaddr_storage lifru_addr;
      struct sockaddr_storage lifru_dstaddr;
      struct sockaddr_storage lifru_broadaddr;
      struct sockaddr_storage lifru_token;    /* With lifr_addrlen */
      struct sockaddr_storage lifru_subnet;   /* With lifr_addrlen */
      int                     lifru_index;    /* interface index */
      uint64_t                lifru_flags;    /* SIOC?LIFFLAGS */
      int                     lifru_metric;
      uint_t                  lifru_mtu;
      int                     lif_muxid[2];   /* mux id's for arp & ip */
      struct lif_nd_req       lifru_nd_req;
      struct lif_ifinfo_req   lifru_ifinfo_req;
      zoneid_t                lifru_zone;     /* SIOC[GS]LIFZONE */
```

```
   } lifr_lifru;

#define lifr_addrlen   lifr_lifru1.lifru_addrlen
#define lifr_ppa       lifr_lifru1.lifru_ppa       /* Driver's ppa */
#define lifr_addr      lifr_lifru.lifru_addr       /* address */
#define lifr_dstaddr   lifr_lifru.lifru_dstaddr
#define lifr_broadaddr lifr_lifru.lifru_broadaddr  /* broadcast addr. */
#define lifr_token     lifr_lifru.lifru_token      /* address token */
#define lifr_subnet    lifr_lifru.lifru_subnet     /* subnet prefix */
#define lifr_index     lifr_lifru.lifru_index      /* interface index */
#define lifr_flags     lifr_lifru.lifru_flags      /* flags */
#define lifr_metric    lifr_lifru.lifru_metric     /* metric */
#define lifr_mtu       lifr_lifru.lifru_mtu        /* mtu */
#define lifr_ip_muxid  lifr_lifru.lif_muxid[0]
#define lifr_arp_muxid lifr_lifru.lif_muxid[1]
#define lifr_nd        lifr_lifru.lifru_nd_req     /* SIOCLIF*ND */
#define lifr_ifinfo    lifr_lifru.lifru_ifinfo_req /* SIOC[GS]LIFLNKINFO */
#define lifr_zone      lifr_lifru.lifru_zone       /* SIOC[GS]LIFZONE */
};
```

`SIOCSLIFADDR` Set interface address.

`SIOCGLIFADDR` Get interface address.

`SIOCSLIFDSTADDR` Set point to point address for interface.

`SIOCGLIFDSTADDR` Get point to point address for interface.

`SIOCSLIFFLAGS` Set interface flags field. If the interface is marked down, any processes currently routing packets through the interface are notified.

`SIOCGLIFFLAGS` Get interface flags.

`SIOCGLIFCONF` Get interface configuration list. This request takes a `lifconf` structure (see below) as a value-result parameter. The `lifc_family` field can be set to `AF_UNSPEC` to retrieve both `AF_INET` and `AF_INET6` interfaces. The `lifc_len` field should be set to the size of the buffer pointed to by `lifc_buf`.

The `lifc_flags` field should usually be set to zero, but callers that need low-level knowledge of the underlying IP interfaces that comprise an IPMP group can set it to `LIFC_UNDER_IPMP` to request that those interfaces be included in the result. Upon success, `lifc_len` contains the length, in bytes, of the array of `lifreq` structures pointed to by `lifc_req`. For each `lifreq` structure, the `lifr_name` and `lifr_addr` fields are valid.

SIOCGLIFNUM	Get number of interfaces. This request returns an integer which is the number of interface descriptions (`struct lifreq`) returned by the SIOCGLIFCONF ioctl (in other words, indicates how large `lifc_len` must be). This request takes a `struct lifnum` (see below) as a value-result parameter. The `lifn_family` field can be set to AF_UNSPEC to count both AF_INET and AF_INET6 interfaces. The `lifn_flags` field should usually be set to zero, but callers that need low-level knowledge of the underlying IP interfaces that comprise an IPMP group can set it to LIFC_UNDER_IPMP to request that those interfaces be included in the count.
SIOCSLIFMTU	Set the maximum transmission unit (MTU) size for interface. Place the request in the `lifru_mtu` field. The MTU can not exceed the physical MTU limitation (which is reported in the DLPI DL_INFO_ACK message).
SIOCGLIFMTU	Get the maximum transmission unit size for interface.
SIOCSLIFMETRIC	Set the metric associated with the interface. The metric is used by routing daemons such as in.routed(1M).
SIOCGLIFMETRIC	Get the metric associated with the interface.
SIOCGLIFMUXID	Get the `ip` and `arp` `muxid` associated with the interface.
SIOCSLIFMUXID	Set the `ip` and `arp` `muxid` associated with the interface.
SIOCGLIFINDEX	Get the interface index associated with the interface.
SIOCSLIFINDEX	Set the interface index associated with the interface.
SIOCGLIFZONE	Get the zone associated with the interface.
SIOCSLIFZONE	Set the zone associated with the interface. Only applies for zones that use the shared-IP instance.
SIOCLIFADDIF	Add a new logical interface on a physical interface using an unused logical interface number.
SIOCLIFREMOVEIF	Remove a logical interface by specifying its IP address or logical interface name.
SIOCSLIFTOKEN	Set the address token used to form IPv6 link-local addresses and for stateless address autoconfiguration.
SIOCGLIFTOKEN	Get the address token used to form IPv6 link-local addresses and for stateless address autoconfiguration.
SIOCSLIFSUBNET	Set the subnet prefix associated with the interface.
SIOCGLIFSUBNET	Get the subnet prefix associated with the interface.

SIOCSLIFLNKINFO	Set link specific parameters for the interface.
SIOCGLIFLNKINFO	Get link specific parameters for the interface.
SIOCLIFDELND	Delete a neighbor cache entry for IPv6.
SIOCLIFGETND	Get a neighbor cache entry for IPv6.
SIOCLIFSETND	Set a neighbor cache entry for IPv6.
SIOCSLIFUSESRC	Set the interface from which to choose a source address. The `lifr_index` field has the interface index corresponding to the interface whose address is to be used as the source address for packets going out on the interface whose name is provided by `lifr_name`. If the `lifr_index` field is set to zero, the previous setting is cleared. See `ifconfig(1M)` for examples of the `usesrc` option.
SIOCGLIFUSESRC	Get the interface index of the interface whose address is used as the source address for packets going out on the interface provided by `lifr_name` field. The value is retrieved in the `lifr_index` field. See `ifconfig(1M)` for examples of the `usesrc` option.
SIOCGLIFSRCOF	Get the interface configuration list for interfaces that use an address hosted on the interface provided by the `lifs_ifindex` field in the `lifsrcof` struct (see below), as a source address. The application sets `lifs_maxlen` to the size (in bytes) of the buffer it has allocated for the data. On return, the kernel sets `lifs_len` to the actual size required. Note, the application could set `lifs_maxlen` to zero to query the kernel of the required buffer size instead of estimating a buffer size. The application tests `lifs_len` <= `lifs_maxlen` -- if that's true, the buffer was big enough and the application has an accurate list. If it is false, it needs to allocate a bigger buffer and try again, and `lifs_len` provides a hint of how big to make the next trial. See `ifconfig(1M)` for examples of the `usesrc` option.
SIOCTONLINK	Test if the address is directly reachable, for example, that it can be reached without going through a router. This request takes an `sioc_addrreq` structure (see below) as a value-result parameter. The `sa_addr` field should be set to the address to test. The `sa_res` field will contain a non-zero value if the address is onlink.
SIOCTMYADDR	Test if the address is assigned to this node. This request takes an `sioc_addrreq` structure (see below) as a value-result parameter. The `sa_addr` field should be set to the address to test. The `sa_res` field will contain a non-zero value if the address is assigned to this node.
SIOCTMYSITE	Test if the address is part of the same site as this node. This request takes an `sioc_addrreq` structure (see below) as a value-result

parameter. The `sa_addr` field should be set to the address to test. The `sa_res` field will contain a non-zero value if the address is in the same site.

The structure used by `SIOCGLIFCONF` has the form:

```
struct lifconf {
   sa_family_t     lifc_family;
   int             lifc_flags;       /* request specific
                                     /* interfaces */
   int             lifc_len;         /* size of assoc. buffer */
   union {
      caddr_t      lifcu_buf;
      struct lifreq *lifcu_req;
   } lifc_lifcu;

#define lifc_buf lifc_lifcu.lifcu_buf   /* buffer address */
#define lifc_req lifc_lifcu.lifcu_req   /* array of structs returned */
};
```

The structure used by `SIOCGLIFNUM` has the form:

```
struct lifnum {
   sa_family_t     lifn_family;
   int             lifn_flags;   /* req. specf. interfaces */
   int             lifn_count;   /* Result */
};
```

The structure used by `SIOCTONLINK`, `SIOCTMYADDR` and `SIOCTMYSITE` has the form:

```
struct sioc_addrreq {
   struct sockaddr_storage sa_addr; /* Address to test */
   int                     sa_res;  /* Result - 0/1 */
};
```

The structure used by `SIOCGLIFSRCOF` has the form:

```
 struct lifsrcof {
     uint_t  lifs_ifindex;   /* addr on this interface */
                             /* used as the src addr */
     size_t  lifs_maxlen;    /* size of buffer: input */
     size_t  lifs_len;       /* size of buffer: output */
     union {
            caddr_t lifsu_buf;
            struct  lifreq *lifsu_req;
   } lifs_lifsu;
#define lifs_buf lifs_lifsu.lifsu_buf /* buffer addr. */
#define lifs_req lifs_lifsu.lifsu_req /* array returned */
};
```

The following `ioctl()` calls are maintained for compatibility but only apply to IPv4 network interfaces, since the data structures are too small to hold an IPv6 address. Unless specified otherwise, the request takes an `ifreq` structure as its parameter. This structure has the form:

```
struct ifreq {
#define IFNAMSIZ    16
 char    ifr_name[IFNAMSIZ];             /* interface name - e.g. "hme0" */
 union {
       struct sockaddr    ifru_addr;
       struct sockaddr    ifru_dstaddr;
       struct sockaddr    ifru_broadaddr;
       short ifru_flags;
       int   ifru_metric;
       int   if_muxid[2];                  /* mux id's for arp and ip */
       int   ifru_index;                    /* interface index */
    } ifr_ifru;

#define ifr_addr      ifr_ifru.ifru_addr       /* address */
#define ifr_dstaddr   ifr_ifru.ifru_dstaddr    /* other end of p-to-p link */
#define ifr_broadaddr ifr_ifru.ifru_broadaddr  /* broadcast address */
#define ifr_flags     ifr_ifru.ifru_flags      /* flags */
#define ifr_index     ifr_ifru.ifru_index      /* interface index  */
#define ifr_metric    ifr_ifru.ifru_metric     /* metric */
};
```

`SIOCSIFADDR` Set interface address.

`SIOCGIFADDR` Get interface address.

`SIOCSIFDSTADDR` Set point to point address for interface.

`SIOCGIFDSTADDR` Get point to point address for interface.

`SIOCSIFFLAGS` Set interface flags field. If the interface is marked down, any processes currently routing packets through the interface are notified.

`SIOCGIFFLAGS` Get interface flags.

`SIOCGIFCONF` Get interface configuration list. This request takes an `ifconf` structure (see below) as a value-result parameter. The `ifc_len` field should be set to the size of the buffer pointed to by `ifc_buf`. Upon success, `ifc_len` will contain the length, in bytes, of the array of `ifreq` structures pointed to by `ifc_req`. For each `ifreq` structure, the `ifr_name` and `ifr_addr` fields are valid. Though IPMP IP interfaces are included in the array, underlying IP interfaces that comprise those IPMP groups are not.

`SIOCGIFNUM` Get number of interfaces. This request returns an integer which is the number of interface descriptions (`struct ifreq`) returned by the `SIOCGIFCONF` ioctl (in other words, indicates how large `ifc_len` must

be). Though IPMP IP interfaces are included in the array, underlying IP interfaces that comprise those IPMP groups are not.

`SIOCSIFMTU` Set the maximum transmission unit (MTU) size for interface. Place the request in the `ifr_metric` field. The MTU has to be smaller than physical MTU limitation (which is reported in the DLPI `DL_INFO_ACK` message).

`SIOCGIFMTU` Get the maximum transmission unit size for interface. Upon success, the request is placed in the `ifr_metric` field.

`SIOCSIFMETRIC` Set the metric associated with the interface. The metric is used by routine daemons such as `in.routed(1M)`.

`SIOCGIFMETRIC` Get the metric associated with the interface.

`SIOCGIFMUXID` Get the `ip` and `arp` `muxid` associated with the interface.

`SIOCSIFMUXID` Set the `ip` and `arp` `muxid` associated with the interface.

`SIOCGIFINDEX` Get the interface index associated with the interface.

`SIOCSIFINDEX` Set the interface index associated with the interface.

The `ifconf` structure has the form:

```
struct ifconf {
 int    ifc_len;                        /* size of assoc. buffer */
 union {
       caddr_t        ifcu_buf;
       struct ifreq  *ifcu_req;
    } ifc_ifcu;

#define ifc_buf   ifc_ifcu.ifcu_buf   /* buffer address */
#define ifc_req   ifc_ifcu.ifcu_req   /* array of structs returned */
};
```

IFF_Flags You can use the `ifconfig(1M)` command to display the `IFF_` flags listed below (with the leading `IFF_` prefix removed). See the `ifconfig(1M)` manpage for a definition of each flag.

```
#define IFF_UP          0x0000000001    /* Address is up */
#define IFF_BROADCAST   0x0000000002    /* Broadcast address valid */
#define IFF_DEBUG       0x0000000004    /* Turn on debugging */
#define IFF_LOOPBACK    0x0000000008    /* Loopback net */

#define IFF_POINTOPOINT 0x0000000010    /* Interface is p-to-p */
#define IFF_NOTRAILERS  0x0000000020    /* Avoid use of trailers */
#define IFF_RUNNING     0x0000000040    /* Resources allocated */
#define IFF_NOARP       0x0000000080    /* No address res. protocol */

#define IFF_PROMISC     0x0000000100    /* Receive all packets */
```

```
#define IFF_ALLMULTI    0x0000000200    /* Receive all multicast pkts */
#define IFF_INTELLIGENT 0x0000000400    /* Protocol code on board */
#define IFF_MULTICAST   0x0000000800    /* Supports multicast */

#define IFF_MULTI_BCAST 0x0000001000    /* Multicast using broadcst. add. */
#define IFF_UNNUMBERED  0x0000002000    /* Non-unique address */
#define IFF_DHCPRUNNING 0x0000004000    /* DHCP controls interface */
#define IFF_PRIVATE     0x0000008000    /* Do not advertise */

#define IFF_NOXMIT      0x0000010000    /* Do not transmit pkts */
#define IFF_NOLOCAL     0x0000020000    /* No address - just on-link subnet */
#define IFF_DEPRECATED  0x0000040000    /* Address is deprecated */
#define IFF_ADDRCONF    0x0000080000    /* Addr. from stateless addrconf */

#define IFF_ROUTER      0x0000100000    /* Router on interface */
#define IFF_NONUD       0x0000200000    /* No NUD on interface */
#define IFF_ANYCAST     0x0000400000    /* Anycast address */
#define IFF_NORTEXCH    0x0000800000    /* Don't xchange rout. info */

#define IFF_IPV4        0x0001000000    /* IPv4 interface */
#define IFF_IPV6        0x0002000000    /* IPv6 interface */
#define IFF_NOFAILOVER  0x0008000000    /* in.mpathd test address */
#define IFF_FAILED      0x0010000000    /* Interface has failed */

#define IFF_STANDBY     0x0020000000    /* Interface is a hot-spare */
#define IFF_INACTIVE    0x0040000000    /* Functioning but not used */
#define IFF_OFFLINE     0x0080000000    /* Interface is offline */
#define IFF_COS_ENABLED 0x0200000000    /* If CoS marking is supported

#define IFF_COS_ENABLED 0x0200000000    /* If CoS marking is supported */
#define IFF_PREFERRED   0x0400000000    /* Prefer as source address */
#define IFF_TEMPORARY   0x0800000000    /* RFC3041 */
#define IFF_FIXEDMTU    0x1000000000    /* MTU set with SIOCSLIFMTU */

#define IFF_VIRTUAL     0x2000000000    /* Cannot send/receive pkts */
#define IFF_DUPLICATE   0x4000000000    /* Local address in use */
#define IFF_IPMP        0x8000000000    /* IPMP IP interface */
```

Errors

`EPERM` Calling process has insufficient privileges.

`ENXIO` The `lifr_name` member of the `lifreq` structure contains an invalid value.

For `SIOCGLIFSRCOF`, the `lifs_ifindex` member of the `lifsrcof` structure contains an invalid value.

For `SIOCSLIFUSESRC`, this error is returned if the `lifr_index` is set to an invalid value.

`EBADADDR` Wrong address family or malformed address.

`EINVAL`	For `SIOCSLIFMTU`, this error is returned when the requested MTU size is invalid. This error indicates the MTU size is greater than the MTU size supported by the DLPI provider or less than `68` (for IPv4) or less than `1280` (for IPv6). For `SIOCSLIFUSESRC`, this error is returned if either the `lifr_index` or `lifr_name` identify interfaces that are already part of an existing IPMP group.
`EEXIST`	For `SIOCLIFADDIF`, this error is returned if the `lifr_name` member in the `lifreq` structure corresponds to an interface that already has the PPA specified by `lifr_ppa` plumbed.

See Also `ifconfig(1M)`, `in.routed(1M)`, `ioctl(2)`, `streamio(7I)`, `arp(7P)`, `dlpi(7P)`, `ip(7P)`, `ip6(7P)`

Name igb – Intel 82575 1Gb PCI Express NIC Driver

Synopsis `/dev/igb*`

Description The `igb` Gigabit Ethernet driver is a multi-threaded, loadable, clonable, GLD-based STREAMS driver supporting the Data Link Provider Interface, `dlpi(7P)`, on Intel 82575 Gigabit Ethernet controllers.

The `igb` driver functions include controller initialization, frame transmit and receive, promiscuous and multicast support, and error recovery and reporting.

The `igb` driver and hardware support auto-negotiation, a protocol specified by the 1000 Base-T standard. Auto-negotiation allows each device to advertise its capabilities and discover those of its peer (link partner). The highest common denominator supported by both link partners is automatically selected, yielding the greatest available throughput, while requiring no manual configuration. The `igb` driver also allows you to configure the advertised capabilities to less than the maximum (where the full speed of the interface is not required), or to force a specific mode of operation, irrespective of the link partner's advertised capabilities.

Application Programming Interface The cloning character-special device, `/dev/igb`, is used to access all Intel 82575 Gigabit devices installed within the system.

The igb driver is managed by the `dladm(1M)` command line utility, which allows VLANs to be defined on top of `igb` instances and for `igb` instances to be aggregated. See `dladm(1M)` for more details.

You must send an explicit DL_ATTACH_REQ message to associate the opened stream with a particular device (PPA). The PPA ID is interpreted as an unsigned integer data type and indicates the corresponding device instance (unit) number. The driver returns an error (DL_ERROR_ACK) if the PPA field value does not correspond to a valid device instance number for the system. The device is initialized on first attach and de-initialized (stopped) at last detach.

The values returned by the driver in the DL_INFO_ACK primitive in response to your DL_INFO_REQ are:

- Maximum SDU is 9000.
- Minimum SDU is 0.
- DLSAP address length is 8.
- MAC type is DL_ETHER.
- SAP (Service Access Point) length value is -2, meaning the physical address component is followed immediately by a 2-byte SAP component within the DLSAP address.
- Broadcast address value is the Ethernet/IEEE broadcast address (FF:FF:FF:FF:FF:FF).

Once in the DL_ATTACHED state, you must send a DL_BIND_REQ to associate a particular SAP with the stream.

Configuration By default, the igb driver performs auto-negotiation to select the link speed and mode. Link speed and mode can be any one of the following, (as described in the *IEEE803.2* standard):

1000 Mbps, full-duplex.

100 Mbps, full-duplex.

100 Mbps, half-duplex.

10 Mbps, full-duplex.

10 Mbps, half-duplex.

The auto-negotiation protocol automatically selects speed (1000 Mbps, 100 Mbps, or 10 Mbps) and operation mode (full-duplex or half-duplex) as the highest common denominator supported by both link partners.

Alternatively, you can set the capabilities advertised by the `igb` device using ndd(1M). The driver supports a number of parameters whose names begin with *adv_* (see below). Each of these parameters contains a boolean value that determines if the device advertises that mode of operation. For example, the *adv_1000fdx_cap* parameter indicates if 1000M full duplex is advertised to link partner. The *adv_autoneg* cap parameter controls whether auto-negotiation is performed. If *adv_autoneg_cap* is set to 0, the driver forces the mode of operation selected by the first non-zero parameter in priority order as shown below:

```
                    (highest priority/greatest throughput)
adv_1000fdx_cap         1000Mbps full duplex
adv_100fdx_cap          100Mpbs full duplex
adv_100hdx_cap          100Mbps half duplex
adv_10fdx_cap           10Mpbs full duplex
adv_10hdx_cap           10Mpbs half duplex
                        (lowest priority/least throughput)
```

All capabilities default to enabled. Note that changing any capability parameter causes the link to go down while the link partners renegotiate the link speed/duplex using the newly changed capabilities.

Files

`/dev/igb*`	Special character device.
`/kernel/drv/igb`	32–bit device driver (x86).
`/kernel/drv/amd64/igb`	64–bit device driver (x86).
`/kernel/drv/sparcv9/igb`	64–bit device driver (SPARC).
`/kernel/drv/igb.conf`	Configuration file.

See Also dladm(1M), ndd(1M), netstat(1M), driver.conf(4), attributes(5), streamio(7I), dlpi(7P),

Writing Device Drivers

STREAMS Programming Guide

Network Interfaces Programmer's Guide

Name ii – Instant Image control device

Description The `ii` device is a control interface for Instant Image devices and controls the Instant Image module through the `ioctl(2)` interface.

Instant Image is a point-in-time volume copy facility for the Solaris operating environment that is administered through the `iiadm(1M)` command. With Instant Image, you can create an independent point-in-time copy of a volume or a master volume-dependent point-in-time view. You can also independently access the master and shadow volume for read and write operations. Instant Image also lets you update the shadow volume from the master volume or restore the master volume from the shadow. (Restore operations to volumes can be full or incremental). Instant Image supports fast volume re-synchronization, letting you create a new point-in-time volume copy by updating the specified volume with only changed data.

To create a shadow volume you need a:

1. Master volume to be shadowed.
2. Shadow volume where the copy will reside. This volume must be equal to or larger than the master volume.
3. Administrative bitmap volume or file for tracking differences between the shadow and master volumes. The administrative bitmap volume or file must be at least 24Kbytes in size and requires 8KBytes for each GByte (or part thereof) of master volume size, plus an additional 8KBytes overhead. For example, to shadow a 3GByte master volume, the administration volume must be 8KBytes + (3 * 8KBytes) =32KBytes in size.

The Instant Image module uses services provided by the SDBC and SD_GEN modules. The SV module is required to present a conventional block device interface to the storage product interface of the Instant Image, SDBC and SD_GEN modules.

When a shadow operation is suspended or resumed, the administration volumes may be stored in permanent SDBC storage or loaded and saved to and from kernel memory. The ii_bitmap variable in the `/kernel/drv/ii.conf` configuration file determines the administration volume storage type. A value of 0 indicates kernel memory, while a value of 1 indicates permanent SDBC storage. If the system is part of a storage products cluster, use the 1 value (permanent storage), otherwise use kernel memory (0 value).

Files

`kernel/drv/ii`	32– bit ELF kernel module (x86).
`/kernel/drv/ii.conf`	Configuration file.

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWiu
Interface Stability	Committed

See Also iiadm(1M), ioctl(2), attributes(5), sv(7D)

Name inet6 – Internet protocol family for Internet Protocol version 6

Synopsis

```
#include <sys/types.h>
#include <netinet/in.h>
```

Description The `inet6` protocol family implements a collection of protocols that are centered around the Internet Protocol version 6 (IPv6) and share a common address format. The `inet6` protocol family can be accessed using the socket interface, where it supports the `SOCK_STREAM`, `SOCK_DGRAM`, and `SOCK_RAW` socket types, or the Transport Level Interface (TLI), where it supports the connectionless (`T_CLTS`) and connection oriented (`T_COTS_ORD`) service types.

Protocols The Internet protocol family for IPv6 included the Internet Protocol Version 6 (IPv6), the Neighbor Discovery Protocol (NDP), the Internet Control Message Protocol (ICMPv6), the Transmission Control Protocol (TCP), and the User Datagram Protocol (UDP).

TCP supports the socket interface's `SOCK_STREAM` abstraction and TLI's `T_COTS_ORD` service type. UDP supports the `SOCK_DGRAM` socket abstraction and the TLI `T_CLTS` service type. See `tcp(7P)` and `udp(7P)`. A direct interface to IPv6 is available using the socket interface. See `ip6(7P)`. ICMPv6 is used by the kernel to handle and report errors in protocol processing. It is also accessible to user programs. See `icmp6(7P)`. NDP is used to translate 128-bit IPv6 addresses into 48–bit Ethernet addresses.

IPv6 addresses come in three types: unicast, anycast, and multicast. A unicast address is an identifier for a single network interface. An anycast address is an identifier for a set of interfaces; a packet sent to an anycast address is delivered to the "nearest" interface identified by that address, pursuant to the routing protocol's measure of distance. A multicast address is an identifier for a set of interfaces; a packet sent to a multicast address is delivered to all interfaces identified by that address. There are no broadcast addresses as such in IPv6; their functionality is superseded by multicast addresses.

For IPv6 addresses, there are three scopes within which unicast addresses are guaranteed to be unique. The scope is indicated by the address prefix. The three varieties are link-local (the address is unique on that physical link), site-local (the address is unique within that site), and global (the address is globally unique).

The three highest order bits for global unicast addresses are set to `001`. The ten highest order bits for site-local addresses are set to `1111 1110 11`. The ten highest order bits for link-local addresses are set to `1111 1110 11`. For multicast addresses, the eight highest order bits are set to `1111 1111`. Anycast addresses have the same format as unicast addresses.

IPv6 addresses do not follow the concept of "address class" seen in IP.

A global unicast address is divided into the following segments:

- The first three bits are the Format Prefix identifying a unicast address.
- The next 13 bits are the Top-Level Aggregation (TLA) identifier. For example, the identifier could specify the ISP.

- The next eight bits are reserved for future use.
- The next 24 bits are the Next-Level Aggregation (NLA) identifier.
- The next 16 bits are the Site-Level Aggregation (SLA) identifier.
- The last 64 bits are the interface ID. This will most often be the hardware address of the link in IEEE EUI-64 format.

Link-local unicast addresses are divided in this manner:

- The first ten bits are the Format Prefix identifying a link-local address.
- The next 54 bits are zero.
- The last 64 bits are the interface ID. This will most often be the hardware address of the link in IEEE EUI-64 format.

Site-local unicast addresses are divided in this manner:

- The first ten bits are the Format Prefix identifying a site-local address.
- The next 38 bits are zero.
- The next 16 bits are the subnet ID.
- The last 64 bits are the interface ID. This will most often be the hardware address of the link in IEEE EUI-64 format.

Addressing IPv6 addresses are sixteen byte quantities, stored in network byte order. The socket API uses the `sockaddr_in6` structure when passing IPv6 addresses between an application and the kernel. The `sockaddr_in6` structure has the following members:

```
sa_familty_t     sin6_family;
in_port_t        sin6_port;
uint32_t         sin6_flowinfo;
struct in6_addr  sin6_addr;
uint32_t         sin6_scope_id;
uint32_t         __sin6_src_id;
```

Library routines are provided to manipulate structures of this form. See `inet(3SOCKET)`.

The `sin6_addr` field of the `sockaddr_in6` structure specifies a local or remote IPv6 address. Each network interface has one or more IPv6 addresses configured, that is, a link-local address, a site-local address, and one or more global unicast IPv6 addresses. The special value of all zeros may be used on this field to test for "wildcard" matching. Given in a `bind(3SOCKET)` call, this value leaves the local IPv6 address of the socket unspecified, so that the socket will receive connections or messages directed at any of the valid IPv6 addresses of the system. This can prove useful when a process neither knows nor cares what the local IPv6 address is, or when a process wishes to receive requests using all of its network interfaces.

The `sockaddr_in6` structure given in the `bind()` call must specify an *in6_addr* value of either all zeros or one of the system's valid IPv6 addresses. Requests to bind any other address will

elicit the error EADDRNOTAVAI. When a connect(3SOCKET) call is made for a socket that has a wildcard local address, the system sets the sin6_addr field of the socket to the IPv6 address of the network interface through which the packets for that connection are routed.

The sin6_port field of the sockaddr_in6 structure specifies a port number used by TCP or UDP. The local port address specified in a bind() call is restricted to be greater than IPPORT_RESERVED (defined in <netinet/in.h>) unless the creating process is running as the super-user, providing a space of protected port numbers. In addition, the local port address cannot be in use by any socket of the same address family and type. Requests to bind sockets to port numbers being used by other sockets return the error EADDRINUSE. If the local port address is specified as 0, the system picks a unique port address greater than IPPORT_RESERVED. A unique local port address is also selected when a socket which is not bound is used in a connect(3SOCKET) or sendto() call. See send(3SOCKET). This allows programs that do not care which local port number is used to set up TCP connections by simply calling socket(3SOCKET) and then connect(3SOCKET), and then sending UDP datagrams with a socket() call followed by a sendto() call.

Although this implementation restricts sockets to unique local port numbers, TCP allows multiple simultaneous connections involving the same local port number so long as the remote IPv6 addresses or port numbers are different for each connection. Programs may explicitly override the socket restriction by setting the SO_REUSEADDR socket option with setsockopt(). See getsockopt(3SOCKET).

In addition, the same port may be bound by two separate sockets if one is an IP socket and the other an IPv6 socket.

TLI applies somewhat different semantics to the binding of local port numbers. These semantics apply when Internet family protocols are used using the TLI.

Source Address Selection

IPv6 source address selection is done on a per destination basis, and utilizes a list of rules from which the best source address is selected from candidate addresses. The candidate set comprises a set of local addresses assigned on the system which are up and not anycast. If just one candidate exists in the candidate set, it is selected.

Conceptually, each selection rule prefers one address over another, or determines their equivalence. If a rule produces a tie, a subsequent rule is used to break the tie.

The sense of some rules may be reversed on a per-socket basis using the IPV6_SRC_PREFERENCES socket option (see ip6(7P)). The flag values for this option are defined in <netinet/in.h> and are referenced in the description of the appropriate rules below.

As the selection rules indicate, the candidate addresses are SA and SB and the destination is D.

Prefer the same address	If SA == D, prefer SA. If SB == D, prefer SB.
Prefer appropriate scope	Here, Scope(X) is the scope of X according to the IPv6 Addressing Architecture.

	If Scope(SA) < Scope(SB): If Scope(SA) < Scope(D), then prefer SB and otherwise prefer SA. If Scope(SB) < Scope(SA): If Scope(SB) < Scope(D), then prefer SA and otherwise prefer SB.
Avoid deprecated addresses	If one of the addresses is deprecated (IFF_DEPRECATED) and the other is not, prefer the one that isn't deprecated.
Prefer preferred addresses	If one of the addresses is preferred (IFF_PREFERRED) and the other is not, prefer the one that is preferred.
Prefer outgoing interface	If one of the addresses is assigned to the interface that will be used to send packets to D and the other is not, then prefer the former.
Prefer matching label	This rule uses labels which are obtained through the IPv6 default address selection policy table. See `ipaddrsel(1M)` for a description of the default contents of the table and how the table is configured. If Label(SA) == Label(D) and Label(SB) != Label(D), then prefer SA. If Label(SB) == Label(D) and Label(SA) != Label(D), then prefer SB.
Prefer public addresses	This rule prefers public addresses over temporary addresses, as defined in *RFC 3041*. Temporary addresses are disabled by default and may be enabled by appropriate settings in `ndpd.conf(4)`. The sense of this rule may be set on a per-socket basis using the IPV6_SRC_PREFERENCES socket option. Passing the flag IPV6_PREFER_SRC_TMP or IPV6_PREFER_SRC_PUBLIC will cause temporary or public addresses to be preferred, respectively, for that particular socket. See `ip6(7P)` for more information about IPv6 socket options.
Use longest matching prefix.	This rule prefers the source address that has the longer matching prefix with the destination. Because this is the last rule and because both source addresses could have equal matching prefixes, this rule does an `xor` of each source address with the destination, then selects the source address with the smaller `xor` value in order to break any potential tie. If SA ^ D < SB ^ D, then prefer SA.

If SB ^ D < SA ^ D, then prefer SB.

Applications can override this algorithm by calling bind(3SOCKET) and specifying an address.

See Also ioctl(2), bind(3SOCKET), connect(3SOCKET), getipnodebyaddr(3SOCKET), getipnodebyname(3SOCKET),getprotobyname(3SOCKET), getservbyname(3SOCKET), getsockopt(3SOCKET), inet(3SOCKET), send(3SOCKET), icmp6(7P), ip6(7P), tcp(7P), udp(7P)

Conta, A. and Deering, S., *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*, RFC 1885, December 1995.

Deering, S. and Hinden, B., *Internet Protocol, Version 6 (IPv6) Specification*, RFC 1883, December 1995.

Hinden, B. and Deering, S., *IP Version 6 Addressing Architecture*, RFC 1884, December 1995.

Draves, R., *RFC 3484, Default Address Selection for IPv6.* The Internet Society. February 2003.

Narten, T., and Draves, R. *RFC 3041, Privacy Extensions for Stateless Address Autoconfiguration in IPv6.* The Internet Society. January 2001.

Notes The IPv6 support is subject to change as the Internet protocols develop. Users should not depend on details of the current implementation, but rather the services exported.

Name inet – Internet protocol family

Synopsis

```
#include <sys/types.h>

#include <netinet/in.h>
```

Description The Internet protocol family implements a collection of protocols which are centered around the Internet Protocol (“IP”) and which share a common address format. The Internet family protocols can be accessed using the socket interface, where they support the `SOCK_STREAM`, `SOCK_DGRAM`, and `SOCK_RAW` socket types, or the Transport Level Interface (TLI), where they support the connectionless (`T_CLTS`) and connection oriented (`T_COTS_ORD`) service types.

Protocols The Internet protocol family is comprised of the Internet Protocol (“IP”), the Address Resolution Protocol (“ARP”), the Internet Control Message Protocol (“ICMP”), the Transmission Control Protocol (“TCP”), and the User Datagram Protocol (“UDP”).

TCP supports the socket interface's `SOCK_STREAM` abstraction and TLI's `T_COTS_ORD` service type. UDP supports the `SOCK_DGRAM` socket abstraction and the TLI `T_CLTS` service type. See `tcp(7P)` and `udp(7P)`. A direct interface to IP is available using both TLI and the socket interface (see `ip(7P)`). ICMP is used by the kernel to handle and report errors in protocol processing. It is also accessible to user programs (see `icmp(7P)`). ARP is used to translate 32-bit IP addresses into 48-bit Ethernet addresses. See `arp(7P)`.

The 32-bit IP address is divided into network number and host number parts. It is frequency-encoded. The most-significant bit is zero in Class A addresses, in which the high-order 8 bits represent the network number. Class B addresses have their high order two bits set to 10 and use the high-order 16 bits as the network number field. Class C addresses have a 24-bit network number part of which the high order three bits are 110. Sites with a cluster of IP networks may chose to use a single network number for the cluster; this is done by using subnet addressing. The host number portion of the address is further subdivided into subnet number and host number parts. Within a subnet, each subnet appears to be an individual network. Externally, the entire cluster appears to be a single, uniform network requiring only a single routing entry. Subnet addressing is enabled and examined by the following `ioctl(2)` commands. They have the same form as the `SIOCSIFADDR` command.

`SIOCSIFNETMASK` Set interface network mask. The network mask defines the network part of the address; if it contains more of the address than the address type would indicate, then subnets are in use.

`SIOCGIFNETMASK` Get interface network mask.

Addressing IP addresses are four byte quantities, stored in network byte order. IP addresses should be manipulated using the byte order conversion routines. See `byteorder(3SOCKET)`.

Addresses in the Internet protocol family use the `sockaddr_in` structure, which has that following members:

```
short    sin_family;
ushort_t sin_port;
struct   in_addr  sin_addr;
char     sin_zero[8];
```

Library routines are provided to manipulate structures of this form; See inet(3SOCKET).

The sin_addr field of the sockaddr_in structure specifies a local or remote IP address. Each network interface has its own unique IP address. The special value INADDR_ANY may be used in this field to effect "wildcard" matching. Given in a bind(3SOCKET) call, this value leaves the local IP address of the socket unspecified, so that the socket will receive connections or messages directed at any of the valid IP addresses of the system. This can prove useful when a process neither knows nor cares what the local IP address is or when a process wishes to receive requests using all of its network interfaces. The sockaddr_in structure given in the bind(3SOCKET) call must specify an in_addr value of either INADDR_ANY or one of the system's valid IP addresses. Requests to bind any other address will elicit the error EADDRNOTAVAIL. When a connect(3SOCKET) call is made for a socket that has a wildcard local address, the system sets the sin_addr field of the socket to the IP address of the network interface that the packets for that connection are routed through.

The sin_port field of the sockaddr_in structure specifies a port number used by TCP or UDP. The local port address specified in a bind(3SOCKET) call is restricted to be greater than IPPORT_RESERVED (defined in <<netinet/in.h>>) unless the creating process is running as the superuser, providing a space of protected port numbers. In addition, the local port address must not be in use by any socket of same address family and type. Requests to bind sockets to port numbers being used by other sockets return the error EADDRINUSE. If the local port address is specified as 0, then the system picks a unique port address greater than IPPORT_RESERVED. A unique local port address is also picked when a socket which is not bound is used in a connect(3SOCKET) or sendto (see send(3SOCKET)) call. This allows programs which do not care which local port number is used to set up TCP connections by simply calling socket(3SOCKET) and then connect(3SOCKET), and to send UDP datagrams with a socket(3SOCKET) call followed by a sendto() call.

Although this implementation restricts sockets to unique local port numbers, TCP allows multiple simultaneous connections involving the same local port number so long as the remote IP addresses or port numbers are different for each connection. Programs may explicitly override the socket restriction by setting the SO_REUSEADDR socket option with setsockopt (see getsockopt(3SOCKET)).

TLI applies somewhat different semantics to the binding of local port numbers. These semantics apply when Internet family protocols are used using the TLI.

See Also ioctl(2), bind(3SOCKET), byteorder(3SOCKET), connect(3SOCKET), gethostbyname(3NSL), getnetbyname(3SOCKET), getprotobyname(3SOCKET), getservbyname(3SOCKET), getsockopt(3SOCKET), send(3SOCKET), socket(3SOCKET), arp(7P), icmp(7P), ip(7P), tcp(7P), udp(7P)

Network Information Center, *DDN Protocol Handbook* (3 vols.), Network Information Center, SRI International, Menlo Park, Calif., 1985.

Notes The Internet protocol support is subject to change as the Internet protocols develop. Users should not depend on details of the current implementation, but rather the services exported.

Name ip6 – Internet Protocol Version 6

Synopsis

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip6.h>

s = socket(AF_INET6, SOCK_RAW, proto);

t = t_open ("/dev/rawip6", O_RDWR);
```

Description The IPv6 protocol is the next generation of the internetwork datagram delivery protocol of the Internet protocol family. Programs may use IPv6 through higher-level protocols such as the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP), or may interface directly to IPv6. See `tcp(7P)` and `udp(7P)`. Direct access may be by means of the socket interface, using a "raw socket," or by means of the Transport Level Interface (TLI). The protocol options and IPv6 extension headers defined in the IPv6 specification may be set in outgoing datagrams.

Application Programming Interface The STREAMS driver `/dev/rawip6` is the TLI transport provider that provides raw access to IPv6.

Raw IPv6 sockets are connectionless and are normally used with the `sendto()` and `recvfrom()` calls (see `send(3SOCKET)` and `recv(3SOCKET)`), although the `connect(3SOCKET)` call may also be used to fix the destination for future datagrams. In this case, the `read(2)` or `recv(3SOCKET)` and `write(2)` or `send(3SOCKET)` calls may be used. Ancillary data may also be sent or received over raw IPv6 sockets using the `sendmsg(3SOCKET)` and `recvmsg(3SOCKET)` system calls.

Unlike raw IP, IPv6 applications do not include a complete IPv6 header when sending; there is no IPv6 analog to the IP `IP_HDRINCL` socket option. IPv6 header values may be specified or received as ancillary data to a `sendmsg(3SOCKET)` or `recvmsg(3SOCKET)` system call, or may be specified as "sticky" options on a per-socket basis by using the `setsockopt(3SOCKET)` system call. Such sticky options are applied to all outbound packets unless overridden by ancillary data. If any ancillary data is specified in a `sendmsg(3SOCKET)` call, all sticky options not explicitly overridden revert to default values for that datagram only; the sticky options persist as set for subsequent datagrams.

Since `sendmsg(3SOCKET)` is not supported for `SOCK_STREAM` upper level protocols such as TCP, ancillary data is unsupported for TCP. Sticky options, however, are supported.

Since `sendmsg(3SOCKET)` is supported for `SOCK_DGRAM` upper level protocols, both ancillary data and sticky options are supported for UDP, ICMP6, and raw IPv6 sockets.

The socket options supported at the IPv6 level are:

`IPV6_BOUND_IF` Limit reception and transmission of packets to this interface. Takes an integer as an argument; the integer is the selected interace index.

`IPV6_UNSPEC_SRC`	Boolean. Allow/disallow sending with a zero source address.
`IPV6_UNICAST_HOPS`	Default hop limit for unicast datagrams. This option takes an integer as an argument. Its value becomes the new default value for `ip6_hops` that IPv6 will use on outgoing unicast datagrams sent from that socket. The initial default is `60`.
`IPV6_CHECKSUM`	Specify the integer offset in bytes into the user data of the checksum location. Does not apply to the ICMP6 protocol. Note: checksums are required for all IPv6 datagrams; this is different from IP, in which datagram checksums were optional. IPv6 will compute the ULP checksum if the value in the checksum field is zero.
`IPV6_SEC_OPT`	Enable or obtain IPsec security settings for this socket. For more details on the protection services of IPsec, see `ipsec(7P)`.
`IPV6_DONTFRAG`	Boolean. Control fragmentation.
`IPV6_USE_MIN_MTU`	Controls whether path MTU discovery is used. If set to 1, path MTU discovery is never used and IPv6 packets are sent with the IPv6 minimum MTU. If set to -1, path MTU discovery is not used for multicast and multicast packets are sent with the IPv6 minimum MTU. If set to 0, path MTU is always performed.
`IPV6_V6ONLY`	Boolean. If set, only V6 packets can be sent or received
`IPV6_SRC_PREFERENCES`	Enable or obtain Source Address Selection rule settings for this socket. For more details on the Source Address Selection rules, see `inet6(7P)`.

The following options are boolean switches controlling the reception of ancillary data:

`IPV6_RECVPKTINFO`	Enable/disable receipt of the index of the interface the packet arrived on, and of the inbound packet's destination address.
`IPV6_RECVHOPLIMIT`	Enable/disable receipt of the inbound packet's current hoplimit.
`IPV6_RECVHOPOPTS`	Enable/disable receipt of the inbound packet's IPv6 hop-by-hop extension header.
`IPV6_RECVDSTOPTS`	Enable/disable receipt of the inbound packet's IPv6 destination options extension header.
`IPV6_RECVRTHDR`	Enable/disable receipt of the inbound packet's IPv6 routing header.

IPV6_RECVRTHDRDSTOPTS	Enable/disable receipt of the inbound packet's intermediate-hops options extension header. This option is obsolete. IPV6_RECVDSTOPTS turns on receipt of both destination option headers.
IPV6_RECVTCLASS	Enable/disable receipt of the traffic class of the inbound packet.
IPV6_RECVPATHMTU	Enable/disable receipt of the path mtu of the inbound packet.

The following options may be set as sticky options with setsockopt(3SOCKET) or as ancillary data to a sendmsg(3SOCKET) system call:

IPV6_PKTINFO	Set the source address and/or interface out which the packet(s) will be sent. Takes a struct in6_pktinfo as the parameter.
IPV6_HOPLIMIT	Set the initial hoplimit for outbound datagrams. Takes an integer as the parameter. Note: This option sets the hoplimit only for ancillary data or sticky options and does not change the default hoplimit for the socket; see IPV6_UNICAST_HOPS and IPV6_MULTICAST_HOPS to change the socket's default hoplimit.
IPV6_NEXTHOP	Specify the IPv6 address of the first hop, which must be a neighbor of the sending host. Takes a struct sockaddr_in6 as the parameter. When this option specifies the same address as the destination IPv6 address of the datagram, this is equivalent to the existing SO_DONTROUTE option.
IPV6_HOPOPTS	Specify one or more hop-by-hop options. Variable length. Takes a complete IPv6 hop-by-hop options extension header as the parameter.
IPV6_DSTOPTS	Specify one or more destination options. Variable length. Takes a complete IPv6 destination options extension header as the parameter.
IPV6_RTHDR	Specify the IPv6 routing header. Variable length. Takes a complete IPv6 routing header as the parameter. Currently, only type 0 routing headers are supported.
IPV6_RTHDRDSTOPTS	Specify one or more destination options for all intermediate hops. May be configured, but will not be applied unless an IPv6 routing header is also configured. Variable length. Takes a complete IPv6 destination options extension header as the parameter.
IPV6_PATHMTU	Get the path MTU associated with a connected socket. Takes a ip6_mtuinfo as the parameter.
IPV6_TCLASS	Set the traffic class associated with outgoing packets. The parameter is an integer. If the parameter is less then -1 or greater then 256,

EINVAL is returned. If the parameter is equal to -1, use the default. If the parameter is between 0 and 255 inclusive, use that value.

The following options affect the socket's multicast behavior:

`IPV6_JOIN_GROUP`	Join a multicast group. Takes a `struct ipv6_mreq` as the parameter; the structure contains a multicast address and an interface index.
`IPV6_LEAVE_GROUP`	Leave a multicast group. Takes a `struct ipv6_mreq` as the parameter; the structure contains a multicast address and an interface index.
`MCAST_JOIN_GROUP`	Functionally equivalent to IPV6_JOIN_GROUP. Takes a `struct group_req` as the parameter. The structure contains a multicast address and an interface index.
`MCAST_BLOCK_SOURCE`	Block multicast packets on a particular multicast group whose source address matches the given source address. The specified group must be joined previously using IPV6_JOIN_GROUP or MCAST_JOIN_GROUP. Takes a `struct group_source_req` as the parameter. The structure contains an interface index, a multicast address, and a source address.
`MCAST_UNBLOCK_SOURCE`	Unblock multicast packets which were previously blocked using MCAST_BLOCK_SOURCE. Takes a `struct group_source_req` as the parameter. The structure contains an interface index, a multicast address, and a source address.
`MCAST_LEAVE_GROUP`	Functionally equivalent to IPV6_LEAVE_GROUP. Takes a `struct group_req` as the parameter. The structure contains a multicast address and an interface index.
`MCAST_JOIN_SOURCE_GROUP`	Begin receiving packets for the given multicast group whose source address matches the specified address. Takes a `struct group_source_req` as the parameter. The structure contains an interface index, a multicast address, and a source address.
`MCAST_LEAVE_SOURCE_GROUP`	Stop receiving packets for the given multicast group whose source address matches the specified address. Takes a `struct group_source_req` as the parameter. The structure contains an interface index, a multicast address, and a source address.

`IPV6_MULTICAST_IF`	The outgoing interface for multicast packets. This option takes an integer as an argument; the integer is the interface index of the selected interface.
`IPV6_MULTICAST_HOPS`	Default hop limit for multicast datagrams. This option takes an integer as an argument. Its value becomes the new default value for `ip6_hops` that IPv6 will use on outgoing multicast datagrams sent from that socket. The initial default is `1`.
`IPV6_MULTICAST_LOOP`	Loopback for multicast datagrams. Normally multicast datagrams are delivered to members on the sending host. Setting the unsigned character argument to 0 will cause the opposite behavior.

The multicast socket options can be used with any datagram socket type in the IPv6 family.

At the socket level, the socket option `SO_DONTROUTE` may be applied. This option forces datagrams being sent to bypass routing and forwarding by forcing the IPv6 hoplimit field to `1`, meaning that the packet will not be forwarded by routers.

Raw IPv6 datagrams can also be sent and received using the TLI connectionless primitives.

Datagrams flow through the IPv6 layer in two directions: from the network *up* to user processes and from user processes *down* to the network. Using this orientation, IPv6 is layered *above* the network interface drivers and *below* the transport protocols such as UDP and TCP. The Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) is logically a part of IPv6. See `icmp6(7P)`.

Unlike IP, IPv6 provides no checksum of the IPv6 header. Also unlike IP, upper level protocol checksums are required. IPv6 will compute the ULP/data portion checksum if the checksum field contains a zero (see `IPV6_CHECKSUM` option above).

IPv6 extension headers in received datagrams are processed in the IPv6 layer according to the protocol specification. Currently recognized IPv6 extension headers include hop-by-hop options header, destination options header, routing header (currently, only type 0 routing headers are supported), and fragment header.

By default, the IPv6 layer will not forward IPv6 packets that are not addressed to it. This behavior can be overridden by using `routeadm(1M)` to enable the ipv6-forwarding option. IPv6 forwarding is configured at boot time based on the setting of `routeadm(1M)`'s ipv6-forwarding option.

For backwards compatibility, IPv6 forwarding can be enabled or disabled using `ndd(1M)`'s ip_forwarding variable. It is set to 1 if IPv6 forwarding is enabled, or 0 if it is disabled.

Additionally, finer-grained forwarding can be configured in IPv6. Each interface can be configured to forward IPv6 packets by setting the IFF_ROUTER interface flag. This flag can be

set and cleared using ifconfig(1M)'s router and –router options. If an interface's IFF_ROUTER flag is set, packets can be forwarded to or from the interface. If it is clear, packets will neither be forwarded from this interface to others, nor forwarded to this interface. Setting the ip6_forwarding variable sets all of the IPv6 interfaces' IFF_ROUTER flags.

For backwards compatibility, each interface creates an `<ifname>ip6_forwarding` `/dev/ip6` variable that can be modified using ndd(1M). An interface's `:ip6_forwarding` `ndd` variable is a boolean variable that mirrors the status of its IFF_ROUTER interface flag. It is set to 1 if the flag is set, or 0 if it is clear. This interface specific `<ifname>:ip6_forwarding` `ndd` variable is obsolete and may be removed in a future release of Solaris. The ifconfig(1M) router and –router interfaces are preferred.

The IPv6 layer will send an ICMP6 message back to the source host in many cases when it receives a datagram that can not be handled. A "`time exceeded`" ICMP6 message will be sent if the `ip6_hops` field in the IPv6 header drops to zero in the process of forwarding a datagram. A "`destination unreachable`" message will be sent by a router or by the originating host if a datagram can not be sent on because there is no route to the final destination; it will be sent by a router when it encounters a firewall prohibition; it will be sent by a destination node when the transport protocol (that is, TCP) has no listener. A "`packet too big`" message will be sent by a router if the packet is larger than the MTU of the outgoing link (this is used for Path MTU Discovery). A "`parameter problem`" message will be sent if there is a problem with a field in the IPv6 header or any of the IPv6 extension headers such that the packet cannot be fully processed.

The IPv6 layer supports fragmentation and reassembly. Datagrams are fragmented on output if the datagram is larger than the maximum transmission unit (MTU) of the network interface. Fragments of received datagrams are dropped from the reassembly queues if the complete datagram is not reconstructed within a short time period.

Errors in sending discovered at the network interface driver layer are passed by IPv6 back up to the user process.

See Also svcs(1), ndd(1M), routeadm(1M), svcadm(1M), read(2), write(2), bind(3SOCKET), connect(3SOCKET), getsockopt(3SOCKET), recv(3SOCKET), recvmsg(3SOCKET), send(3SOCKET), sendmsg(3SOCKET), setsockopt(3SOCKET), defaultrouter(4), smf(5), icmp6(7P), if_tcp(7P), ipsec(7P), inet6(7P), routing(7P), tcp(7P), udp(7P)

Deering, S. and Hinden, B. *RFC 2460, Internet Protocol, Version 6 (IPv6) Specification*. The Internet Society. December, 1998.

Stevens, W., and Thomas, M. *RFC 2292, Advanced Sockets API for IPv6*. Network Working Group. February 1998.

Diagnostics A socket operation may fail with one of the following errors returned:

`EPROTONOSUPPORT` Unsupported protocol (for example, IPPROTO_RAW.)

EACCES	A `bind()` operation was attempted with a "reserved" port number and the effective user ID of the process was not the privileged user.
EADDRINUSE	A `bind()` operation was attempted on a socket with a network address/port pair that has already been bound to another socket.
EADDRNOTAVAIL	A `bind()` operation was attempted for an address that is not configured on this machine.
EINVAL	A `sendmsg()` operation with a non-NULL `msg_accrights` was attempted.
EINVAL	A `getsockopt()` or `setsockopt()` operation with an unknown socket option name was given.
EINVAL	A `getsockopt()` or `setsockopt()` operation was attempted with the IPv6 option field improperly formed; an option field was shorter than the minimum value or longer than the option buffer provided; the value in the option field was invalid.
EISCONN	A `connect()` operation was attempted on a socket on which a `connect()` operation had already been performed, and the socket could not be successfully disconnected before making the new connection.
EISCONN	A `sendto()` or `sendmsg()` operation specifying an address to which the message should be sent was attempted on a socket on which a `connect()` operation had already been performed.
EMSGSIZE	A `send()`, `sendto()`, or `sendmsg()` operation was attempted to send a datagram that was too large for an interface, but was not allowed to be fragmented (such as broadcasts).
ENETUNREACH	An attempt was made to establish a connection via `connect()`, or to send a datagram by means of `sendto()` or `sendmsg()`, where there was no matching entry in the routing table; or if an ICMP "`destination unreachable`" message was received.
ENOTCONN	A `send()` or `write()` operation, or a `sendto()` or `sendmsg()` operation not specifying an address to which the message should be sent, was attempted on a socket on which a `connect()` operation had not already been performed.
ENOBUFS	The system ran out of memory for fragmentation buffers or other internal data structures.
ENOMEM	The system was unable to allocate memory for an IPv6 socket option or other internal data structures.

ENOPROTOOPT	An IP socket option was attempted on an IPv6 socket, or an IPv6 socket option was attempted on an IP socket.
ENOPROTOOPT	Invalid socket type for the option.

Notes Applications using the sockets API must use the Advanced Sockets API for IPv6 (*RFC 2292*) to see elements of the inbound packet's IPv6 header or extension headers.

The `ip6` service is managed by the service management facility, `smf(5)`, under the service identifier:

```
svc:/network/initial:default
```

Administrative actions on this service, such as enabling, disabling, or requesting restart, can be performed using `svcadm(1M)`. The service's status can be queried using the `svcs(1)` command.

Name ip, IP – Internet Protocol

Synopsis

```
#include <sys/socket.h>

#include <netinet/in.h>

s = socket(AF_INET, SOCK_RAW, proto);

t = t_open ("/dev/rawip", O_RDWR);
```

Description IP is the internetwork datagram delivery protocol that is central to the Internet protocol family. Programs may use IP through higher-level protocols such as the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP), or may interface directly to IP. See `tcp(7P)` and `udp(7P)`. Direct access may be by means of the socket interface, using a "raw socket," or by means of the Transport Level Interface (TLI). The protocol options defined in the IP specification may be set in outgoing datagrams.

Packets sent to or from this system may be subject to IPsec policy. See `ipsec(7P)` for more information.

Application Programming Interface The STREAMS driver `/dev/rawip` is the TLI transport provider that provides raw access to IP.

Raw IP sockets are connectionless and are normally used with the `sendto()` and `recvfrom()` calls (see `send(3SOCKET)` and `recv(3SOCKET)`), although the `connect(3SOCKET)` call may also be used to fix the destination for future datagram. In this case, the `read(2)` or `recv(3SOCKET)` and `write(2)` or `send(3SOCKET)` calls may be used. If *proto* is `IPPROTO_RAW` or `IPPROTO_IGMP`, the application is expected to include a complete IP header when sending. Otherwise, that protocol number will be set in outgoing datagrams and used to filter incoming datagrams and an IP header will be generated and prepended to each outgoing datagram. In either case, received datagrams are returned with the IP header and options intact.

If an application uses `IP_HDRINCL` and provides the IP header contents, the IP stack does not modify the following supplied fields under any conditions: Type of Service, DF Flag, Protocol, and Destination Address. The IP Options and IHL fields are set by use of `IP_OPTIONS`, and `Total Length` is updated to include any options. Version is set to the default. Identification is chosen by the normal IP ID selection logic. The source address is updated if none was specified and the TTL is changed if the packet has a broadcast destination address. Since an applicaton cannot send down fragments (as IP assigns the IP ID), Fragment Offset is always 0. The IP Checksum field is computed by IP. None of the data beyond the IP header are changed, including the application-provided transport header.

The socket options supported at the IP level are:

`IP_OPTIONS` IP options for outgoing datagrams. This socket option may be used to set IP options to be included in each outgoing datagram. IP options to be sent are set with `setsockopt()` (see `getsockopt(3SOCKET)`). The `getsockopt(3SOCKET)` call returns the IP options set in the last `setsockopt()` call. IP options on

received datagrams are visible to user programs only using raw IP sockets. The format of IP options given in `setsockopt()` matches those defined in the IP specification with one exception: the list of addresses for the source routing options must include the first-hop gateway at the beginning of the list of gateways. The first-hop gateway address will be extracted from the option list and the size adjusted accordingly before use. IP options may be used with any socket type in the Internet family.

`IP_SEC_OPT` Enable or obtain IPsec security settings for this socket. For more details on the protection services of IPsec, see `ipsec(7P)`.

`IP_ADD_MEMBERSHIP` Join a multicast group.

`IP_DROP_MEMBERSHIP` Leave a multicast group.

`IP_BOUND_IF` Limit reception and transmission of packets to this interface. Takes an integer as an argument. The integer is the selected interface index.

The following options take `in_pktinfo_t` as the parameter:

`IP_PKTINFO`
Set the source address and/or transmit interface of the packet(s). Note that the IP_BOUND_IF socket option takes precedence over the interface index passed in IP_PKTINFO.

```
struct in_pktinfo {
   unsigned int ipi_ifindex;/* send/recv interface index */
   struct in_addr ipi_spec_dst;/* matched source addr. */
   struct in_addr ipi_addr;/* src/dst addr. in IP hdr */
} in_pktinfo_t;
```

When passed in (on transmit) via ancillary data with IP_PKTINFO, ipi_spec_dst is used as the source address and ipi_ifindex is used as the interface index to send the packet out.

`IP_RECVPKTINFO`
Enable/disable receipt of the index of the interface the packet arrived on, the local address that was matched for reception, and the inbound packet's actual destination address. Takes boolean as the parameter. Returns struct in_pktinfo_t as ancillary data.

The following options take a `struct ip_mreq` as the parameter. The structure contains a multicast address which must be set to the `CLASS-D` IP multicast address and an interface address. Normally the interface address is set to `INADDR_ANY` which causes the kernel to choose the interface on which to join.

`IP_BLOCK_SOURCE` Block multicast packets whose source address matches the given source address. The specified group must be joined previously using IP_ADD_MEMBERSHIP or MCAST_JOIN_GROUP.

IP_UNBLOCK_SOURCE	Unblock (begin receiving) multicast packets which were previously blocked using IP_BLOCK_SOURCE.
IP_ADD_SOURCE_MEMBERSHIP	Begin receiving packets for the given multicast group whose source address matches the specified address.
IP_DROP_SOURCE_MEMBERSHIP	Stop receiving packets for the given multicast group whose source address matches the specified address.

The following options take a `struct ip_mreq_source` as the parameter. The structure contains a multicast address (which must be set to the CLASS-D IP multicast address), an interface address, and a source address.

MCAST_JOIN_GROUP	Join a multicast group. Functionally equivalent to IP_ADD_MEMBERSHIP.
MCAST_BLOCK_SOURCE	Block multicast packets whose source address matches the given source address. The specified group must be joined previously using IP_ADD_MEMBERSHIP or MCAST_JOIN_GROUP.
MCAST_UNBLOCK_SOURCE	Unblock (begin receiving) multicast packets which were previously blocked using MCAST_BLOCK_SOURCE.
MCAST_LEAVE_GROUP	Leave a multicast group. Functionally equivalent to IP_DROP_MEMBERSHIP.
MCAST_JOIN_SOURCE_GROUP	Begin receiving packets for the given multicast group whose source address matches the specified address.
MCAST_LEAVE_SOURCE_GROUP	Stop receiving packets for the given multicast group whose source address matches the specified address.

The following options take a struct `group_req` or struct `group_source_req` as the parameter. The ‘`group_req` structure contains an interface index and a multicast address which must be set to the CLASS-D multicast address. The `group_source_req` structure is used for those options which include a source address. It contains an interface index, multicast address, and source address.

IP_MULTICAST_IF	The outgoing interface for multicast packets. This option takes a `struct in_addr` as an argument, and it selects that interface for outgoing IP multicast packets. If the address specified is `INADDR_ANY`, it uses the unicast routing table to select the outgoing interface (which is the default behavior).
IP_MULTICAST_TTL	Time to live for multicast datagrams. This option takes an unsigned character as an argument. Its value is the TTL that IP uses on outgoing multicast datagrams. The default is `1`.

`IP_MULTICAST_LOOP` Loopback for multicast datagrams. Normally multicast datagrams are delivered to members on the sending host (or sending zone). Setting the unsigned character argument to 0 causes the opposite behavior, meaning that when multiple zones are present, the datagrams are delivered to all zones except the sending zone.

`IP_RECVIF` Receive the inbound interface index.

`IP_TOS` This option takes an integer argument as its input value. The least significant 8 bits of the value are used to set the Type Of Service field in the IP header of the outgoing packets.

`IP_DONTFRAG` This option controls whether IP will allow fragmentation both locally (fragmenting the packets before sending them out on the wire), and in the network (whether or not the `Don't Fragment` flag will be set in the IPv4 header). Setting the option to any `non-zero` value disables fragmentation. Setting the option to zero enables fragmentation. When fragmentation is disabled then IP will not create any Path MTU state on behalf of this socket.

`IP_NEXTHOP` This option specifies the address of the onlink nexthop for traffic originating from that socket. It causes the routing table to be bypassed and outgoing traffic is sent directly to the specified nexthop. This option takes an ipaddr_t argument representing the IPv4 address of the nexthop as the input value. The IP_NEXTHOP option takes precedence over IPOPT_LSRR. IP_BOUND_IF and SO_DONTROUTE take precedence over IP_NEXTHOP. This option has no meaning for broadcast and multicast packets. The application must ensure that the specified nexthop is alive. An application may want to specify the IP_NEXTHOP option on a TCP listener socket only for incoming requests to a particular IP address. In this case, it must avoid binding the socket to INADDR_ANY and instead must bind the listener socket to the specific IP address. In addition, typically the application may want the incoming and outgoing interface to be the same. In this case, the application must select a suitable nexthop that is onlink and reachable via the desired interface and do a setsockopt (IP_NEXTHOP) on it. Then it must bind to the IP address of the desired interface. Setting the IP_NEXTHOP option requires the PRIV_SYS_NET_CONFIG privilege.

The multicast socket options (IP_MULTICAST_IF, IP_MULTICAST_TTL, IP_MULTICAST_LOOP and IP_RECVIF) can be used with any datagram socket type in the Internet family.

At the socket level, the socket option `SO_DONTROUTE` may be applied. This option forces datagrams being sent to bypass routing and forwarding by forcing the IP Time To Live field to 1, meaning that the packet will not be forwarded by routers.

Raw IP datagrams can also be sent and received using the TLI connectionless primitives.

Datagrams flow through the IP layer in two directions: from the network *up* to user processes and from user processes *down* to the network. Using this orientation, IP is layered *above* the network interface drivers and *below* the transport protocols such as UDP and TCP. The Internet Control Message Protocol (ICMP) is logically a part of IP. See `icmp(7P)`.

IP provides for a checksum of the header part, but not the data part, of the datagram. The checksum value is computed and set in the process of sending datagrams and checked when receiving datagrams.

IP options in received datagrams are processed in the IP layer according to the protocol specification. Currently recognized IP options include: security, loose source and record route (LSRR), strict source and record route (SSRR), record route, and internet timestamp.

By default, the IP layer will not forward IPv4 packets that are not addressed to it. This behavior can be overridden by using `routeadm(1M)` to enable the ipv4-forwarding option. IPv4 forwarding is configured at boot time based on the setting of `routeadm(1M)`'s ipv4-forwarding option.

For backwards compatibility, IPv4 forwarding can be enabled or disabled using `ndd(1M)`'s ip_forwarding variable. It is set to 1 if IPv4 forwarding is enabled, or 0 if it is disabled.

Additionally, finer-grained forwarding can be configured in IP. Each interface can be configured to forward IP packets by setting the IFF_ROUTER interface flag. This flag can be set and cleared using `ifconfig(1M)`'s router and router options. If an interface's IFF_ROUTER flag is set, packets can be forwarded to or from the interface. If it is clear, packets will neither be forwarded from this interface to others, nor forwarded to this interface. Setting the ip_forwarding variable sets all of the IPv4 interfaces' IFF_ROUTER flags.

For backwards compatibility, each interface creates an `<ifname>:ip_forwarding` `/dev/ip` variable that can be modified using `ndd(1M)`. An interface's `:ip_forwarding` `ndd` variable is a boolean variable that mirrors the status of its IFF_ROUTER interface flag. It is set to 1 if the flag is set, or 0 if it is clear. This interface specific `<ifname>` `:ip_forwarding` `ndd` variable is obsolete and may be removed in a future release of Solaris. The `ifconfig(1M)` router and –router interfaces are preferred.

The IP layer sends an ICMP message back to the source host in many cases when it receives a datagram that can not be handled. A “time exceeded” ICMP message is sent if the “time to live” field in the IP header drops to zero in the process of forwarding a datagram. A “destination unreachable” message is sent if a datagram can not be forwarded because there is no route to the final destination, or if it can not be fragmented. If the datagram is addressed to

the local host but is destined for a protocol that is not supported or a port that is not in use, a destination unreachable message is also sent. The IP layer may send an ICMP "source quench" message if it is receiving datagrams too quickly. ICMP messages are only sent for the first fragment of a fragmented datagram and are never returned in response to errors in other ICMP messages.

The IP layer supports fragmentation and reassembly. Datagrams are fragmented on output if the datagram is larger than the maximum transmission unit (MTU) of the network interface. Fragments of received datagrams are dropped from the reassembly queues if the complete datagram is not reconstructed within a short time period.

Errors in sending discovered at the network interface driver layer are passed by IP back up to the user process.

Multi-Data Transmit allows more than one packet to be sent from the IP module to another in a given call, thereby reducing the per-packet processing costs. The behavior of Multi-Data Transmit can be overrideen by using `ndd(1M)` to set the `/dev/ip` variable, ip_multidata_outbound to 0. Note, the IP module will only initiate Multi-Data Transmit if the network interface driver supports it.

Packet Events Through the netinfo framework, this driver provides the following packet events:

Physical in	Packets received on a network interface from an external source.
Physical out	Packets to be sent out a network interface.
Forwarding	Packets being forwarded through this host to another network.
loopback in	Packets that have been sent by a local application to another.
loopback out	Packets about to be received by a local application from another.

Currently, only a single function may be registered for each event. As a result, if the slot for an event is already occupied by someone else, a second attempt to register a callback fails.

To receive packet events in a kernel module, it is first necessary to obtain a handle for either IPv4 or IPv6 traffic. This is achieved by passing NHF_INET or NHF_INET6 through to a net_protocol_lookup() call. The value returned from this call must then be passed into a call to net_register_hook(), along with a description of the hook to add. For a description of the structure passed through to the callback, please see `hook_pkt_event(9S)`. For IP packets, this structure is filled out as follows:

hpe_ifp	Identifier indicating the inbound interface for packets received with the `physical in` event.
hpe_ofp	Identifier indicating the outbound interface for packets received with the `physical out` event.
hpe_hdr	Pointer to the start of the IP header (not the ethernet header).

hpe_mp	Pointer to the start of the mblk_t chain containing the IP packet.
hpe_mb	Pointer to the mblk_t with the IP header in it.

Network Interface Events

In addition to events describing packets as they move through the system, it is also possible to receive notification of events relating to network interfaces. These events are all reported back through the same callback. The list of events is as follows:

plumb	A new network interface has been instantiated.
unplumb	A network interface is no longer associated with this protocol.
up	At least one logical interface is now ready to receive packets.
down	There are no logical interfaces expecting to receive packets.
address change	An address has changed on a logical interface.

See Also

`ifconfig(1M)`, `routeadm(1M)`, `ndd(1M)`, `read(2)`, `write(2)`, `bind(3SOCKET)`, `connect(3SOCKET)`, `getsockopt(3SOCKET)`, `recv(3SOCKET)`, `send(3SOCKET)`, `defaultrouter(4)`, `icmp(7P)`, `if_tcp(7P)`, `inet(7P)`, `ip6(7P)`, `ipsec(7P)`, `routing(7P)`, `tcp(7P)`, `udp(7P)`, `net_hook_register(9F)`, `hook_pkt_event(9S)`

Braden, R., *RFC 1122, Requirements for Internet Hosts – Communication Layers*, Information Sciences Institute, University of Southern California, October 1989.

Postel, J., *RFC 791, Internet Protocol – DARPA Internet Program Protocol Specification*, Information Sciences Institute, University of Southern California, September 1981.

Diagnostics

A socket operation may fail with one of the following errors returned:

`EACCES`	A `bind()` operation was attempted with a "reserved" port number and the effective user ID of the process was not the privileged user. Setting the IP_NEXTHOP was attempted by a process lacking the PRIV_SYS_NET_CONFIG privilege.
`EADDRINUSE`	A `bind()` operation was attempted on a socket with a network address/port pair that has already been bound to another socket.
`EADDRNOTAVAIL`	A `bind()` operation was attempted for an address that is not configured on this machine.
`EINVAL`	A `sendmsg()` operation with a non-NULL `msg_accrights` was attempted.
`EINVAL`	A `getsockopt()` or `setsockopt()` operation with an unknown socket option name was given.

EINVAL	A `getsockopt()` or `setsockopt()` operation was attempted with the IP option field improperly formed; an option field was shorter than the minimum value or longer than the option buffer provided.
EISCONN	A `connect()` operation was attempted on a socket on which a `connect()` operation had already been performed, and the socket could not be successfully disconnected before making the new connection.
EISCONN	A `sendto()` or `sendmsg()` operation specifying an address to which the message should be sent was attempted on a socket on which a `connect()` operation had already been performed.
EMSGSIZE	A `send()`, `sendto()`, or `sendmsg()` operation was attempted to send a datagram that was too large for an interface, but was not allowed to be fragmented (such as broadcasts).
ENETUNREACH	An attempt was made to establish a connection by means of `connect()`, or to send a datagram by means of `sendto()` or `sendmsg()`, where there was no matching entry in the routing table; or if an ICMP "destination unreachable" message was received.
ENOTCONN	A `send()` or `write()` operation, or a `sendto()` or `sendmsg()` operation not specifying an address to which the message should be sent, was attempted on a socket on which a `connect()` operation had not already been performed.
ENOBUFS	The system ran out of memory for fragmentation buffers or other internal data structures.
ENOBUFS	`SO_SNDBUF` or `SO_RCVBUF` exceeds a system limit.
EINVAL	Invalid length for `IP_OPTIONS`.
EHOSTUNREACH	Invalid address for `IP_MULTICAST_IF`.
	Invalid (offlink) nexthop address for IP_NEXTHOP.
EINVAL	Not a multicast address for `IP_ADD_MEMBERSHIP` and `IP_DROP_MEMBERSHIP`.
EADDRNOTAVAIL	Bad interface address for `IP_ADD_MEMBERSHIP` and `IP_DROP_MEMBERSHIP`.
EADDRINUSE	Address already joined for `IP_ADD_MEMBERSHIP`.
ENOENT	Address not joined for `IP_DROP_MEMBERSHIP`.
ENOPROTOOPT	Invalid socket type.
EPERM	No permissions.

Notes Raw sockets should receive ICMP error packets relating to the protocol; currently such packets are simply discarded.

Users of higher-level protocols such as TCP and UDP should be able to see received IP options.

Name ipge – PCI-E Gigabit-Ethernet device driver for Intel 82571–based ethernet controller.

Synopsis `/dev/ipge`

Description The `ipge` Sun Gigabit-Ethernet driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface, `dlpi(7P)`. Multiple PCI-E based adapters installed within the system are supported by the driver. The `ipge` driver provides basic support for the PCI-E–based Ethernet hardware and is used to handle pci8086,105e (PCI-E) devices. Functions include chip initialization, frame transmit and receive, multicast and promiscuous support, and error recovery and reporting. The PCI-E device provides 1000BASE-SX networking interfaces using PCI-E ASIC, external SERDES and fiber optical transceiver, or 10/100/1000BASE-T using a PCI-E ASIC attached to a GMII twisted pair copper transceiver, or 10/100BASE-T using a PCI-E ASIC attached to a MII twisted pair copper transceiver.

The 1000Base-SX standard specifies an "auto-negotiation" protocol to automatically select the mode of operation. In addition to duplex mode of operation, the MAC controller can auto-negotiate for *IEEE 802.3x* frame based flow control capabilities. The PCI-E PCS is capable of doing auto-negotiation with the remote-end of the link (link partner) and receives the capabilities of the remote end. It selects the highest common denominator mode of operation based on the priorities and also supports forced-mode of operation, in which the driver selects the mode of operation.

Application Programming Interface The `/dev/ipge` cloning character-special device is used to access all `ipge` controllers installed within the system.

ipge and Dlpi The `ipge` driver is a "style 2" Data Link Service provider. All M_PROTO and M_PCPROTO type messages are interpreted as DLPI primitives. Valid DLPI primitives are defined in `<sys/dlpi.h>`. Refer to `dlpi(7P)` for more information. An explicit DL_ATTACH_REQ message by a DLS user is required to associate an opened stream to a particular device (ppa). The ppa ID is interpreted as an `unsigned long` data type and indicates the corresponding device instance (unit) number. An error (DL_ERROR_ACK) is returned by the driver if the ppa field value does not correspond to a valid device instance number in the system. The device is initialized on first attach and un-initialized (stopped) during last detach.

The values returned by the driver in the DL_INFO_ACK primitive in response to the DL_INFO_REQ are as follows:

- Maximum SDU is `1500`. (ETHERMTU - defined in `<sys/ethernet>`.
- Minimum SDU is `0`.
- DSLAP address length is `8` bytes.
- MAC type is `DL_ETHER`.
- SAP length value is -2 meaning the physical address component is followed immediately by a 2-byte sap component within the DLSAP address.

- Service mode is DL_CLDLS.
- No optional quality of service (QOS) support is currently included and the QOS fields are 0.
- Provider style is DL_STYLE2.
- Version is DL_VERSION_2.
- Broadcast address value is Ethernet/IEEE broadcast address (FF:FF:FF:FF:FF:FF).

Once in the DL_ATTACHED state, you must send a DL_BIND_REQ to associate a particular SAP (Service Access Point) with the stream. The `ipge` driver interprets the sap field within the DL_BIND_REQ as an Ethernet "type," meaning valid values for the sap field are in the [0-0xFFFF] range. Only one Ethernet type can be bound to the stream at any time.

When you select a sap with a value of 0, the receiver is in "802.3 mode." All frames received from the media having a "type" field in the range [0-1500] are assumed to be *802.3* frames and are routed up all open streams which are bound to sap value 0. If more than one stream is in "802.3 mode" then the frame is duplicated and routed up multiple streams as DL_UNITDATA_IND messages.

During transmission, the driver checks if either the sap value is 0 or destination type field is in the range [0-1500]. If true, the driver sets MAC frame header length field with the length of DL_UNITDATA_REQ message blocks, excluding initial M_PROTO message block, and transmits as *802.3* frames.

The `ipge` driver DLSAP address format consists of the 6 byte physical (Ethernet) address component followed by 2 byte sap (type) component producing an 8 byte DLSAP address. Applications should not hard code to this particular implementation-specific DLSAP address format but use information returned in the DL_INFO_ACK primitive to compose and decompose DLSAP addresses. The sap length, full DLSAP length, and sap/physical ordering are included within the DL_INFO_ACK. The physical address length can be computed by subtracting the sap length from the full DLSAP address length or by issuing the DL_PHYS_ADDR_REQ to obtain the current physical address associated with the stream.

Once the stream is in the DL_BOUND state, you may begin transmitting by sending DL_UNITDATA_REQ messages to the driver.

During receive, the driver routes all received Ethernet frames as DL_UNITDATA_IND messages to all open and bound streams whose sap matches the Ethernet type of the received frame. Received Ethernet frames are duplicated and routed up multiple open streams if necessary. The DLSAP address contained within the DL_UNITDATA_REQ and DL_UNITDATA_IND messages consists of both the sap (type) and physical (Ethernet) components.

dlpi Primitives In addition to the mandatory connectionless DLPI messages, the driver supports the primitives described below.

The DL_ENABMULTI_REQ and DL_DISABMULTI_REQ primitives enable/disable reception of individual multicast group addresses. A set of multicast addresses may be iteratively created and modified on a per-stream basis using DL_ENABMULTI_REQ and DL_DISABMULTI_REQ. DL_ENABMULTI_REQ and DL_DISABMULTI_REQ are accepted by the driver in any state following DL_ATTACHED state.

The DL_PROMISCON_REQ and DL_PROMISCOFF_REQ primitives with the DL_PROMISC_PHYS flag set in `dl_level` field enables/disables reception of all promiscuous mode frames on the media including frames generated by the local host. When used with the DL_PROMISC_SAP flag set, this enables/disables reception of all sap (Ethernet type) values. When used with the DL_PROMISC_MULTI flag set, this enables/disables reception of all multicast group addresses. The effect of each is always on a per-stream basis and independent of the other sap and physical level configurations on this stream or other streams.

The DL_PHYS_ADDR_REQ primitive returns the 6 octet Ethernet address currently associated (attached) to the stream in the DL_PHYS_ADDR_ACK primitive. DL_PHYS_ADDR_REQ is valid only in states following a successful DL_ATTACH_REQ.

The DL_SET_PHYS_ADDR_REQ primitive changes the 6 octet Ethernet address currently associated (attached) to the stream. The credentials of the process which originally opened the stream must be superuser or EPERM is returned in the DL_ERROR_ACK. Because it affects all current and future streams attached to the device, the DL_SET_PHYS_ADDR_REQ is destructive. An M_ERROR is sent up all other streams attached to the device when DL_SET_PHYS_ADDR_REQ is successful on the stream. Once changed, all streams subsequently opened and attached to the device obtain the new physical address. Once changed, the physical address remains until DL_SET_PHYS_ADDR_REQ is used to change the physical address again or the system is rebooted, whichever occurs first.

Configuration By default, the `ipge` driver performs auto-negotiation to select the link speed and mode. Link speed and mode can be any one of the following, (as described in the *IEEE803.2* standard):

- 1000 Mbps, full-duplex
- 1000 Mbps, half-duplex
- 100 Mbps, full-duplex
- 100 Mbps, half-duplex
- 10 Mbps, full-duplex
- 10 Mbps, half-duplex

The auto-negotiation protocol automatically selects:

- Speed (1000 Mbps, 100 Mbps, or 10 Mbps)
- Operation mode (full-duplex or half-duplex)

The auto-negotiation protocol:

Gets all the modes of operation supported by the link partner.
Advertises its capabilities to the link partner.
Selects the highest common denominator mode of operation based on the priorities

The PCI-E hardware is capable of all of the operating modes listed above, when by default, auto-negotiation is used to bring up the link and select the common mode of operation with the link partner. Forced-mode of operation is supported (in which the driver selects the mode of operation and the flow control capabilities) using the `ndd(1M)` utility.

Parameters The `ipge` driver enables setting/getting of various parameters for the PCI-E device. The parameter list includes current transceiver status, current link status, interpacket gap, PCS capabilities and link partner capabilities. PCS capabilities consist of two sets: one reflects the capabilities of the hardware (which are read-only (RO) parameters), while the second reflects the values you choose and is used in speed selection. At boot time, thse two sets of capabilities are identical. By default, the link partner capabilities are read only and cannot be modified.

Files

`/dev/ipge`	Character special device.
`/kernel/drv/ipge.conf`	System wide default device driver properties.

See Also `ndd(1M)`, `netstat(1M)`, `driver.conf(4)`, `dlpi(7P)`

Name ipgpc – IP Generic Packet Classifier

Description The IP Generic Packet Classifier (`ipgpc`) module provides packet classification at the Solaris IP layer. `ipgpc` is an implementation of the Multi-Field (MF) classifier as described in *RFC2475: An Architecture for Differentiated Services*.

The classifier is configured, at startup or dynamically, using a set of "filters." Filters describe selectors that are matched against input packets that are processed by the classifier. Some selectors feature exact matching data points, while others utilize non-exact or wildcard data points.

Each filter is associated with a class describing the next actions to process a packet. There is a many-to-one (M-to-1) mapping relationship between filters and a class. Additionally, each class is aware of which filters are associated with it. A class is configured with a class name and a next action.

Unlike traditional classifiers used in edge routers, `ipgpc` is designed for a host or server device. A host-based classifier provides access to more resources and data than edge routers. User, project, and interface information are available at the host.

Statistics The `ipgpc` module exports global and per-class statistics (available through kstat:)

Global statistics:

```
module: ipgpc                          instance:<action id>
name:   ipgpc global stats             class: <action name>
        crtime
        snaptime
        nbytes                         <number of classified bytes>
        nclasses                       <number of classes>
        nfilters                       <number of filters>
        npackets                       <number of classified packets>
        epackets                       <number of packets in error>
```

Per-class statistics:

```
module: ipgpc                          instance:<action id>
name:   <class name>                   class: <action name>
        crtime
        snaptime
        last match                     <time of last match>
        nbytes                         <number of classified bytes>
        npackets                       <number of classified packets>
```

Files `/kernel/ipp/sparcv9/ipgpc` 64-bit module (SPARC only.)

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWqos

See Also ipqosconf(1M), dlcosmk(7ipp), dscpmk(7ipp), flowacct(7ipp), ipqos(7ipp), tokenmt(7ipp), tswtclmt(7ipp)

RFC 2475, An Architecture for Differentiated Services S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss — The Internet Society, 1998

Name ipnat – IP Filter/NAT module interface

Description The `ipnat` device provides interfaction with the NAT features of the Solaris IPFilter.

Application Programming Interface The NAT features programming model is a component of the Solaris IP Filter and is accessed via the NAT device file `/dev/ipnat`. Opening the device for reading or writing determines which ioctl calls can be successfully made.

ioctls The caller must construct a `ipfobj` structure when issuing a `SIOCGNATL` or `SIOCSTPUT`. The `ipfobj` structure is then passed to the ioctl call and is filled out with ipfo_type set to `IPFOBJ_value`. `IPFOBJ_ value` provides a matching name for the structure, while ipfo_size is set to the total size of the structure being passed and ipfo_ptr is set to the structure address. The ipfo_rev structure should be set to the current value of IPFILTER_VERSION, while ipfo_offset and ipfo_xxxpad should be set to 0.

```
/*
    * Structure used with SIOCGNATL/SIOCSTPUT.
    */
   /*
    * Object structure description.  For passing through in ioctls.
    */
   typedef struct  ipfobj  {
        u_32_t  ipfo_rev;         /* IPFilter version (IPFILTER_VERSION) */
        u_32_t  ipfo_size;        /* size of object at ipfo_ptr */
        void    *ipfo_ptr;        /* pointer to object */
        int     ipfo_type;        /* type of object being pointed to */
        int     ipfo_offset;      /* bytes from ipfo_ptr where to start */
        u_char  ipfo_xxxpad[32];  /* reserved for future use */
   } ipfobj_t;

   #define IPFILTER_VERSION        4010901 /* IPFilter version */
   #define IPFOBJ_NATSAVE          8       /* struct nat_save */
   #define IPFOBJ_NATLOOKUP        9       /* struct natlookup */
```

The following ioctl() calls may be used to manipulate the ipnat sub-system inside of ipf. Note that the ipnat driver only accept calls from applications using the same data model as the kernel. In other words, 64–bit kernels can only accept calls from 64–bit applications. Calls from 32–bit applications fail with `EINVAL`.

`SIOCSTLCK` Set or clear the NAT lock to prevent table updates attributable to packet flow-through.

`SIOCGNATL` Search the NAT table for the rdr entry that matches the fields in the natlookup structure. The caller must populate the structure with the address/port information of the accepted TCP connection (nl_inip, nl_inport) and the address/port information of the peer (nl_outip, nl_outport). The nl_flags field must have the IPN_TCP option set. All other fields must be set to 0. If the call

succeeds, nl_realip and nl_realport are set to the real destination address and port, respectively. The nl_inport and nl_outport fields must be in host byte order.

If IPN_FINDFORWARD is set in nl_flags, a check is made to see if it is possible to create an outgoing NAT session by checking if a packet coming from (nl_realip,nl_realport) and destined for (nl_outip,nl_outport) can be translated. If translation is possible, the flag remains set, otherwise it is cleared in the structure returned to the caller.

```
/*
 * Structure used with SIOCGNATL.
 */
typedef struct natlookup {
     i6addr_t  nl_inipaddr;
     i6addr_t  nl_outipaddr;
     i6addr_t  nl_realipaddr;
     int       nl_v;
     int       nl_flags;
     u_short   nl_inport;
     u_short   nl_outport;
     u_short   nl_realport;
} natlookup_t

#define nl_inip       nl_inipaddr.in4
#define nl_outip      nl_outipaddr.in4
#define nl_realip     nl_realipaddr.in4
#define nl_inip6      nl_inipaddr.in6
#define nl_outip6     nl_outipaddr.in6
#define nl_realip6    nl_realipaddr.in6

 /*
  * Accepted values for nl_flags
  */
 #define   IPN_TCP         0x00001
 #define   IPN_FINDFORWARD 0x400000
```

SIOCSTPUT Move a NAT mapping structure from user space into the kernel. This ioctl is used by ipfs(1M) to restore NAT sessions saved in /var/db/ipf/ipnat.ipf. The nat_save structure must have its ipn_nat and ipn_ipnat structures filled out correctly. Fields not assigned a value must be initialised to 0. All pointer fields are adjusted, as appropriate, once the structure is passed into the kernel and none are preserved.

To create a translation, the following fields must be set:

Interface name - The interface name on which the host is to be exited must be set in nat_ifnames[0].

Local IP address and port number - The connection's local IP address and port number are stored in network byte order using nat_inip/nat_inport.

Destination address/port - The destination address/port are stored in nat_oip/nat_oport.

Target address/port - The translation's target address/port is stored in nat_outip/nat_outport.

The caller must also precalculate the checksum adjustments necessary to complete the translation and store those values in nat_sumd (delta required for TCP header) and nat_ipsumd (delta required for IP header).

```
/*
      * Structures used with SIOCSTPUT.
      */
     typedef struct  nat_save    {
          void    *ipn_next;
          struct  nat     ipn_nat;
          struct  ipnat   ipn_ipnat;
          struct  frentry ipn_fr;
          int     ipn_dsize;
          char    ipn_data[4];
     } nat_save_t;

     typedef struct  nat     {
          ipfmutex_t      nat_lock;
          struct  nat     *nat_next;
          struct  nat     **nat_pnext;
          struct  nat     *nat_hnext[2];
          struct  nat     **nat_phnext[2];
          struct  hostmap *nat_hm;
          void            *nat_data;
          struct  nat     **nat_me;
          struct  ipstate *nat_state;
          struct  ap_session      *nat_aps;
          frentry_t       *nat_fr;
          struct  ipnat   *nat_ptr;
          void            *nat_ifps[2];
          void            *nat_sync;
          ipftqent_t      nat_tqe;
          u_32_t          nat_flags;
```

```
        u_32_t          nat_sumd[2];
        u_32_t          nat_ipsumd;
        u_32_t          nat_mssclamp;
        i6addr_t        nat_inip6;
        i6addr_t        nat_outip6;
        i6addr_t        nat_oip6;
        U_QUAD_T        nat_pkts[2];
        U_QUAD_T        nat_bytes[2];
        union   {
             udpinfo_t       nat_unu;
             tcpinfo_t       nat_unt;
             icmpinfo_t      nat_uni;
             greinfo_t       nat_ugre;
        } nat_un;
        u_short         nat_oport;
        u_short         nat_use;
        u_char          nat_p;
        int             nat_dir;
        int             nat_ref;
        int             nat_hv[2];
        char            nat_ifnames[2][LIFNAMSIZ];
        int             nat_rev;
           int             nat_v;
} nat_t;

#define nat_inip        nat_inip6.in4
#define nat_outip       nat_outip6.in4
#define nat_oip         nat_oip6.in4
#define nat_inport      nat_un.nat_unt.ts_sport
#define nat_outport     nat_un.nat_unt.ts_dport
/*
 * Values for nat_dir
 */
#define NAT_INBOUND     0
#define NAT_OUTBOUND    1
/*
 * Definitions for nat_flags
 */
#define NAT_TCP         0x0001  /* IPN_TCP */
```

Examples The following example shows how to prepare and use `SIOCSTPUT` to insert a NAT session directly into the table. Note that the usual TCP/IP code is omitted is this example.

In the code segment below, incoming_fd is the TCP connection file descriptor that is accepted as part of the redirect process, while remote_fd is the outgoing TCP connection to the remote server being translated back to the original IP address/port pair.

Note – The following ipnat headers must be included before you can use the code shown in this example:

```
#include <netinet/in.h>
#include <arpa/inet.h>
#include <net/if.h>
#include <netinet/ipl.h>
#include <netinet/ip_compat.h>
#include <netinet/ip_fil.h>
#include <netinet/ip_nat.h>
#include <string.h>
#include <fcntl.h>
```

Note – In the example below, various code fragments have been excluded to enhance clarity.

```
int
     translate_connection(int incoming_fd)
     {
          struct sockaddr_in usin;
          struct natlookup nlp;
          struct nat_save ns;
          struct ipfobj obj;
          struct nat *nat;
          int remote_fd;
          int nat_fd;
          int onoff;

          memset(&ns, 0, sizeof(ns));
          nat = &ns.ipn_nat

          namelen = sizeof(usin);
          getsockname(remote_fd, (struct sockaddr *)&usin, &namelen);

          namelen = sizeof(sin);
          getpeername(incoming_fd, (struct sockaddr *) &sin, &namelen);

          namelen = sizeof(sloc);
          getsockname(incoming_fd, (struct sockaddr *) &sloc, &namelen);

          bzero((char *) &obi, sizeof(obj));
          obj.ipfo_rev = IPFILTER_VERSION;
          obj.ipfo_size = sizeof(nlp);
          obj.ipfo_ptr = &nip;
          obj.ipfo_type = IPFOBJ_NATLOOKUP;

          /*
           * Build up the NAT natlookup structure.
           */
```

```
bzero((char *) &nlp, sizeof(nlp));
nlp.nl_outip = sin.sin_addr;
nlp.nl_inip = sloc.sin_addr;
nlp.nl_flags = IPN_TCP;
nlp.nl_outport = ntohs(sin.sin_port);
nlp.nl_inport = ntohs(sloc.sin_port);

/*
 * Open the NAT device and lookup the mapping pair.
 */
nat_fd = open(IPNAT_NAME, O_RDWR);
if (ioctl(nat_fd, SIOCGNATL, &obj) != 0)
     return -1;

nat->nat_inip = usin.sin_addr;
nat->nat_outip = nlp.nl_outip;
nat->nat_oip = nlp.nl_realip;

sum1 = LONG_SUM(ntohl(usin.sin_addr.s_addr)) +
       ntohs(usin.sin_port);
sum2 = LONG_SUM(ntohl(nat->nat_outip.s_addr)) +
       ntohs(nlp.nl_outport);
CALC_SUMD(sum1, sum2, sumd);
nat->nat_sumd[0] = (sumd & 0xffff) + (sumd >> 16);
nat->nat_sumd[1] = nat->nat_sumd[0];

sum1 = LONG_SUM(ntohl(usin.sin_addr.s_addr));
sum2 = LONG_SUM(ntohl(nat->nat_outip.s_addr));
CALC_SUMD(sum1, sum2, sumd);
nat->nat_ipsumd = (sumd & 0xffff) + (sumd >> 16);

nat->nat_inport = usin.sin_port;
nat->nat_outport = nlp.nl_outport;
nat->nat_oport = nlp.nl_realport;

nat->nat_flags = IPN_TCPUDP;

/*
 * Prepare the ipfobj structure, accordingly.
 */
bzero((char *)&obi, sizeof(obj));
obj.ipfo_rev = IPFILTER_VERSION;
obj.ipfo_size = sizeof(*nsp);
obj.ipfo_ptr = nsp;
obj.ipfo_type = IPFOBJ_NATSAVE;

onoff = 1;
```

```
        if (ioctl(nat_fd, SIOCSTPUT, &obj) != 0)
             fprintf(stderr, "Error occurred\n");

        return connect(rem_fd, (struct sockaddr ) &usin, sizeof(usin));
}
```

Errors

EPERM	The device has been opened for reading only. To succeed, the ioctl call must be opened for both reading and writing. The call may be returned if it is privileged and the calling process did not assert {`PRIV_SYS_NET_CONFIG`} in the effective set.
ENOMEM	More memory was allocated than the kernel can provide. The call may also be returned if the application inserts a NAT entry that exceeds the hash bucket chain's maximum length.
EFAULT	The calling process specified an invalid pointer in the ipfobj structure.
EINVAL	The calling process detected a parameter or field set to an unacceptable value.
EEXIST	The calling process, via `SIOCSTPUT`, attempted to add a NAT entry that already exists in the NAT table.
ESRCH	The calling process called `SIOCSTPUT` before setting the SI_NEWFR flag and providing a pointer in the nat_fr field that cannot be found in the current rule set.
EACESS	The calling process issued a `SIOCSTPUT` before issuing a SIOCSTLCK.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed

See Also `ipfs(1M)`, `ipnat(1M)`, `ioctl(2)`, `attributes(5)`

Name ipnet, lo0 – ipnet device driver

Synopsis `/dev/ipnet/*, /dev/lo0`

Description The `ipnet` device driver creates, removes and manages nodes in the `/dev/ipnet/` namespace.

A node is created in `/dev/ipnet` for every IP interface on the system, including interfaces that exist only in software and for which there is no hardware device. The `ipnet` device also provides access to all IP traffic to and from the system. To provide access to packets that are internally looped-back in IP, the `ipnet` driver creates a `/dev/lo0` DLPI device.

Application Programming Interface Device nodes created in `/dev/ipnet` are DLPI style-1 devices. All `M_PROTO` and `M_PCPROTO`–type messages are interpreted as DLPI primitives. Because the device is read-only and packets can only be observed by opening them, the following subset of DLPI primitives is supported:

```
DL_INFO_REQ
DL_BIND_REQ
DL_UNBIND_REQ
DL_PROMISCON_REQ
DL_PROMISCOFF_REQ
DLIOCRAW
```

The values returned by the driver in the `DL_INFO_ACK` primitive in response to the `DL_INFO_REQ` are:

- Maximum SDU is `INT_MAX`
- Minimum SDU is `0`.
- `DLSAP` address length is `2`.
- MAC type is `DL_IPNET`.
- `SAP` length value is 2.
- Service mode is `DL_CLDLS`.
- No optional quality of service (QOS) support is provided. Accordingly, the QOS fields are 0.
- Provider style is `DL_STYLE1`.
- Version is `DL_VERSION_2`.

The `/dev/ipnet/*` and `/dev/lo0` devices only accept `DL_BIND_REQ` requests for SAPs 4 (IPv4 packets), 6 (IPv6 packets), or 0 (all IP packets). `DL_BIND_REQ` requests for other SAP values result in a `DL_ERROR_ACK` of `DL_BADSAP`.

ipnet primitives

For `/dev/ipnet/*` devices, the `DL_PROMISCON_REQ` and `DL_PROMISCOFF_REQ` primitives with the `DL_PROMISC_PHYS` flag set in the `dl_level` field enables/disables the reception of all packets. When disabled, only packets with addresses matching any of the configured addresses on the IP interface are received. When used with the `DL_PROMISC_MULTI` flag set, reception of all multicast group addresses can be enabled/disabled. `DL_PROMISC_PHYS` and `DL_PROMISC_MULTI` have no effect for `/dev/lo0`. When the `DL_PROMISC_SAP` flag is set, reception of all IPv4/IPv6 can be enabled/disabled.

The `DLIOCRAW` ioctl is supported but has no effect on the data returned from the device.

The `DL_IOC_IPNET_INFO` ioctl enables/disables the inclusion of a `dl_ipnetinfo_t` structure that is prepended to the IP header when receiving packet data. When enabled, a non-zero integer is returned reflecting the current `DL_IOC_IPNET_INFO` version. The `dl_ipnetinfo_t` data structure is defined in `<sys/dlpi.h>` and includes the following fields:

```
uint8_t  dli_version;  /* DL_IPNETINFO_* version */
uint8_t  dli_ipver;    /* packet IP header version */
uint16_t dli_len;      /* length of dl_ipnetinfo_t */
uint64_t dli_srczone;  /* packet source zone ID (if any) */
uint64_t dli_dstzone;  /* packet dest zone ID (if any) */
```

The current `dli_version` is 1. To robustly support future `dl_ipnetinfo_t` versions, consumers should check that `dli_version` is a value they recognize, and must use the `dli_len` field to advance past the `dl_ipnetinfo_t` header.

Files

`/dev/ipnet/*, /dev/lo0` Special character devices.

`/kernel/drv/ipnet.conf` Configuration file.

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWckr
Architecture	SPARC, x86
Interface Stability	Committed

See Also

`attributes(5)`, `dlpi(7P)`

Name ipqos – IP Quality of Service

Description `ipqos` is an implementation of the Differentiated Services model defined in *RFC2475: An Architecture for Differentiated Services*, which defines the following entities: multi-field classifier, meter, marker, and dropper. The Solaris implementation of `ipqos` adds a flow accounting entity.

These entities can be combined into processing paths that constitute a series of actions that are performed on groups of flows. The classifier groups together flows and directs them in a given processing path. Classifier configuration and path construction are achieved using the `ipqosconf(1M)` command.

A summary of the `ipqos` entities follows. For more information, refer to the corresponding man page for each entity.

ipgpc	An implementation of the classifier defined in the model. `ipgpc` has been extended and is able to select traffic based on IP header parameters, user id, project id, interface name, interface group and direction.
tokenmt, tswtclmt	These modules implement different metering algorithms. `tokenmt` implements both *RFC2697: A Single Rate Three Color Marker* and *RFC 2698: A Two Rate Three Color Marker*. *tswtclmt* implements *RFC2859: A Time Sliding Window Three Color Marker*. These modules only implement the metering functions defined in the RFCs.
dlcosmk	A marker entity that allows the setting of the user priority field of Ethernet frames as defined in the *IEEE 802.1D* specification. `dlcosmk` is only available with VLAN capable network interfaces.
dscpmk	A marker entity that enables the setting of the Differentiated Services Code Point Value in the IP header as defined in *RFC 2474: Definition of the Differentiated Services Field (DS Field)* in the IPv4 and IPv6 headers.
flowacct	An accounting module that utilizes the Solaris extended accounting facility. `flowacct` logs all flows with parameters used to build a charge back mechanism.

Statistics `ipqos` modules export statistics through the `kstat` facility. Exported statistics contain the following common parameters:

module	module name
instance	dynamic parameter identifying a specific instance
name	a string for global statistics (for example, ipgpc global stat) or a class name for per-class statistics for a classifier action

To verify classifier configuration, generate traffic for each of the configured classes and check that the statistic counters for the expected class are increased. If you're unsure about the parameters for your traffic, you can use `snoop(1M)` to determine them.

Some actions have the instance id of the next configured action in their statistics. This instance id can be used to follow the action processing path. Instance id's -1 and -2 are the built-in actions continue and drop, respectively.

Examples:

To retrieve all statistics for ipgpc:

```
kstat -m ipgpc
```

To retrieve statistics for the class http:

```
kstat -m ipgpc -c http
```

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWqos (32–bit) SUNWqosx (64–bit)

See Also

`ipqosconf(1M)`, `dlcosmk(7ipp)`, `dscpmk(7ipp)`, `flowacct(7ipp)`, `ipgpc(7ipp)`, `tokenmt(7ipp)`, `tswtclmt(7ipp)`

RFC 2475, An Architecture for Differentiated Services S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss — The Internet Society, 1998

RFC 2474, Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers K. Nichols, S. Blake, F. Baker, D. Black — The Internet Society, 1998

RFC 2697, A Single Rate Three Color Marker J. Heinanen, R. Guerin — The Internet Society, 1999

RFC 2698, A Two Rate Three Color Marker J. Heinanen, R. Guerin — The Internet Society, 1999

RFC 2859, A Time Sliding Window Three Colour Marker (TSWTCM) W. Fang, N. Seddigh, B. Nandy — The Internet Society, 2000

Name iprb – Intel 82557, 82558, 82559–controlled network interface controllers

Synopsis `/dev/iprb`

Description The `iprb` Ethernet driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface, `dlpi(7P)`, over Intel D100 82557, 82558, and 82559 controllers. Multiple 82557, 82558, and 82559 controllers installed within the system are supported by the driver. The `iprb` driver provides basic support for the 82557, 82558, and 82559 hardware. Functions include chip initialization, frame transmit and receive, multicast support, and error recovery and reporting.

Application Programming Interface

The cloning, character-special device `/dev/iprb` is used to access all 82557, 82558, and 82559 devices installed within the system.

iprb and DLPI

The `iprb` driver is dependent on `/kernel/misc/gld`, a loadable kernel module that provides the `iprb` driver with the `DLPI` and `STREAMS` functionality required of a LAN driver. See `gld(7D)` for more details on the primitives supported by the driver.

The values returned by the driver in the `DL_INFO_ACK` primitive in response to the `DL_INFO_REQ` from the user are as follows:

- Maximum SDU is `1500` (`ETHERMTU`).
- Minimum SDU is `0`. The driver will pad to the mandatory 60-octet minimum packet size.
- The `dlsap` address length is `8`.
- MAC type is `DL_ETHER`.
- The `sap` length value is –2, meaning the physical address component is followed immediately by a 2-byte `sap` component within the `DLSAP` address.
- Broadcast address value is Ethernet/IEEE broadcast address (`FF:FF:FF:FF:FF:FF`).

Known Problems And Limitations

x86 based systems with the Intel EtherExpress PRO/100B or the Intel EtherExpress PRO/100+ might hang when the interface is brought down at the very instant that a packet is being received. To avoid this, wait until the system is experiencing light or no network traffic before bringing the interface down.

Early versions of the firmware on Intel EtherExpress PRO/100+ and Intel PRO/100+ Management adapters do not support PXE network boot on Solaris systems. Upgrade the firmware if the version is lower than 078. PXE firmware versions are expressed as three-digit build numbers. The build number is typically displayed by the firmware during boot. If the PXE build number is not displayed during boot, change the system BIOS or adapter BIOS configuration to display PXE messages during boot.

Files

`iprb`	Device special file
`/kernel/drv/iprb.conf`	`iprb` configuration file
`<sys/stropts.h>`	`stropts` network header file
`<sys/ethernet.h>`	Ethernet network header file

`<sys/dlpi.h>` `dlpi` network header file

`<sys/gld.h>` `gld` network header file

The `iprb.conf` configuration file options include:

`-TxURRetry` Default: 3

Allowed Values: *0*, *1*, *2*, *3*

Sets the number of retransmissions. Modified when tuning performance.

`-MWIEnable` Default: *0* (Disable)

Allowed Values: *0* (Disable), *1* (Enable)

Should only be set for 82558 adapters and systems in which the PCI bus supports Memory Write & Invalidate operations. Can improve the performance for some configurations.

`-FlowControl` Default: *0* (Disable)

Allowed Values: *0* (Disable), *1* (Enable)

Setting this value can improve the performance for some configurations

`-CollisionBackOffModification` Default: *0* (Disable)

Allowed Values: *0* (Disable), *1* (Enable)

Setting this value can improve the performance for some configurations

`-PhyErrataFrequency` Default: *0* (Disable)

Allowed Values: *0* (Disable), *10* (Enable)

If you have problems establishing links with cables length = 70 Ft, set this field to *10*

`-CpuCycleSaver` Default: *0*

Allowed Values: *1* through *FFFFh*

Reasonable Values: *200h* through *800h*

The CPUSaver algorithm improves the system's P/E ratio by reducing the number of interrupts generated by the card. The algorithm bundles multiple receive frames together, then generates a single interrupt for the bundle. Because the microcode does not support run-time

configuration, configuration must be done prior to the micro code being loaded into the chip. Changing this value from its default means that the driver will have to be unloaded and loaded for the change to take affect. Setting the CpuCycleSaver option to *0* prevents the algorithm from being used. Because it varies for different network environments, the optimal value for this parameter is impossible to predict. Accordingly, developers should run tests to determine the effect that changing this value has on bandwidth and CPU utilization.

`-ForceSpeedDuplex` Default: *5* (Auto-negotiate)

Allowed Values: *4* (100 FDX)

3 (100 HDX)

2 (10 FDX)

1 (10 HDX)

Specify the speed and duplex mode for each instance.

Example: ForceSpeedDuplex=5,4;

Sets `iprb0` to autonegotiate and `iprb1` to 100 FDX.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86

See Also `attributes(5)`, `dlpi(7P)`, `gld(7D)`

Name ipsec – Internet Protocol Security Architecture

Description The IP Security Architecture (IPsec) provides protection for IP datagrams. The protection can include confidentiality, strong integrity of the data, partial sequence integrity (replay protection), and data authentication. IPsec is performed inside the IP processing, and it can be applied with or without the knowledge of an Internet application.

IPsec applies to both IPv4 and IPv6. See `ip(7P)` and `ip6(7P)`.

Protection Mechanisms IPsec provides two mechanisms for protecting data. The Authentication Header (AH) provides strong integrity, replay protection, and data authentication. AH protects as much of the IP datagram as it can. AH cannot protect fields that change nondeterministically between sender and receiver.

The Encapsulating Security Payload (ESP) provides confidentiality over what it encapsulates, as well as the services that AH provides, but only over that which it encapsulates. ESP's authentication services are optional, which allow ESP and AH to be used together on the same datagram without redundancy.

Authentication and encryption algorithms are used for IPsec. Authentication algorithms produce an integrity checksum value or "digest"based on the data and a key. Encryption algorithms operate on data in units of a "block size".

NAT Traversal IPsec's ESP can also encapsulate itself in UDP if IKE (see `in.iked(1M)`) discovers a Network Address Translator (NAT) between two communicating endpoints.

A UDP socket can be specified as a NAT-Traversal endpoint. See `udp(7P)` for details.

Security Associations AH and ESP use Security Associations (SA). SA's are entities that specify security properties from one host to another. Two communicating machines require two SAs (at a minimum) to communicate securely. However, communicating machines that use multicast can share the same multicast SA. SAs are managed through the `pf_key(7P)` interface. For IPv4, automatic SA management is available through the Internet Key Exchange (IKE), as implemented by `in.iked(1M)`. A command-line front-end is available by means of `ipseckey(1M)`. An IPsec SA is identified by a tuple of <AH or ESP, destination IP address, and SPI>. The Security Parameters Index (SPI) is an arbitrary 32-bit value that is transmitted on the wire with an AH or ESP packet. See `ipsecah(7P)` or `ipsecesp(7P)` for an explanation about where the SPI falls in a protected packet.

Protection Policy and Enforcement Mechanisms Mechanism and policy are separate. The policy for applying IPsec is enforced on a system-wide or per-socket level. Configuring system-wide policy and per-tunnel policy (see Transport Mode and Tunnel Mode sections) is done via the `ipsecconf(1M)` command. Configuring per-socket policy is discussed later in this section.

System-wide IPsec policy is applied to incoming and outgoing datagrams. Some additional rules can be applied to outgoing datagrams because of the additional data known by the system. Inbound datagrams can be accepted or dropped. The decision to drop or accept an

inbound datagram is based on several criteria which sometimes overlap or conflict. Conflict resolution is resolved by which rule is parsed first, with one exception: if a policy entry states that traffic should bypass all other policy, it is automatically be accepted. Outbound datagrams are sent with or without protection. Protection may (or may not) indicate specific algorithms. If policy normally would protect a datagram, it can be bypassed either by an exception in system-wide policy or by requesting a bypass in per-socket policy.

Intra-machine traffic policies are enforced, but actual security mechanisms are not applied. Instead, the outbound policy on an intra-machine packet translates into an inbound packet with those mechanisms applied.

IPsec policy is enforced in the `ip(7P)` driver. Several `ndd` tunables for `/dev/ip` affect policy enforcement, including:

icmp_accept_clear_messages	If equal to 1 (the default), allow certain cleartext icmp messages to bypass policy. For ICMP echo requests ("ping"messages), protect the response like the request. If zero, treat icmp messages like other IP traffic.
igmp_accept_clear_messages	If 1, allow inbound cleartext IGMP messages to bypass IPsec policy.
pim_accept_clear_messages	If 1, allow inbound cleartext PIM messages to bypass IPsec policy.
ipsec_policy_log_interval	IPsec logs policy failures and errors to `/var/adm/messages`. To prevent syslog from being overloaded, the IPsec kernel modules limit the rate at which errors can be logged. You can query/set ipsec_policy_log_interval using `ndd(1M)`. The value is in milliseconds. Only one message can be logged per interval.

Transport Mode and Tunnel Mode

If IPsec is used on a tunnel, Tunnel Mode IPsec can be used to protect distinct flows within a tunnel or to cause packets that do not match per-tunnel policy to drop. System-wide policy is always Transport Mode. A tunnel can use Transport Mode IPsec or Tunnel Mode IPsec.

Per-Socket Policy

The `IP_SEC_OPT` or `IPV6_SEC_OPT` socket option is used to set per-socket IPsec policy. The structure used for an `IP_SEC_OPT` request is:

```
typedef struct ipsec_req {
    uint_t      ipsr_ah_req;           /* AH request */
    uint_t      ipsr_esp_req;          /* ESP request */
    uint_t      ipsr_self_encap_req;   /* Self-Encap request */
    uint8_t     ipsr_auth_alg;         /* Auth algs for AH */
    uint8_t     ipsr_esp_alg;          /* Encr algs for ESP */
    uint8_t     ipsr_esp_auth_alg;     /* Auth algs for ESP */
} ipsec_req_t;
```

The IPsec request has fields for both AH and ESP. Algorithms may or may not be specified. The actual request for AH or ESP services can take one of the following values:

`IPSEC_PREF_NEVER` Bypass all policy. Only the superuser may request this service.

`IPSEC_PREF_REQUIRED` Regardless of other policy, require the use of the IPsec service.

The following value can be logically ORed to an `IPSEC_PREF_REQUIRED` value:

`IPSEC_PREF_UNIQUE` Regardless of other policy, enforce a unique SA for traffic originating from this socket.

In the event IP options not normally encapsulated by ESP need to be, the `ipsec_self_encap_req` is used to add an additional IP header outside the original one. Algorithm values from <`net/pfkeyv2.h`> are as follows:

`SADB_AALG_MD5HMAC` Uses the MD5-HMAC (*RFC 2403*) algorithm for authentication.

`SADB_AALG_SHA1HMAC` Uses the SHA1-HMAC (*RFC 2404)* algorithm for authentication.

`SADB_EALG_DESCBC` Uses the DES (*RFC 2405*) algorithm for encryption.

`SADB_EALG_3DESCBC` Uses the Triple DES (*RFC 2451*) algorithm for encryption.

`SADB_EALG_BLOWFISH` Uses the Blowfish (*RFC 2451*) algorithm for encryption.

`SADB_EALG_AES` Uses the Advanced Encryption Standard algorithm for encryption.

`SADB_AALG_SHA256HMAC`
`SADB_AALG_SHA384HMAC`
`SADB_AALG_SHA512HMAC` Uses the SHA2 hash algorithms with HMAC (*RFC 4868*)for authentication.

An application should use either the `getsockopt`(3SOCKET) or the `setsockopt`(3SOCKET) call to manipulate IPsec requests. For example:

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <net/pfkeyv2.h>   /* For SADB_*ALG_* */
/* .... socket setup skipped */
rc = setsockopt(s, IPPROTO_IP, IP_SEC_OPT,
   (const char *)&ipsec_req, sizeof (ipsec_req_t));
```

Security While IPsec is an effective tool in securing network traffic, it will not make security problems disappear. Security issues beyond the mechanisms that IPsec offers may be discussed in similar "Security" or "Security Consideration" sections within individual reference manual pages.

While a non-root user cannot bypass IPsec, a non-root user can set policy to be different from the system-wide policy. For ways to prevent this, consult the `ndd`(1M) variables in `/dev/ip`.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed

See Also in.iked(1M), ipsecconf(1M), ipseckey(1M), ndd(1M), getsockopt(3SOCKET), setsockopt(3SOCKET), attributes(5), inet(7P), ip(7P), ip6(7P), ipsecah(7P), ipsecesp(7P), pf_key(7P), udp(7P)

Kent, S., and Atkinson, R., *RFC 2401, Security Architecture for the Internet Protocol*, The Internet Society, 1998.

Kent, S. and Atkinson, R., *RFC 2406, IP Encapsulating Security Payload (ESP)*, The Internet Society, 1998.

Madson, C., and Doraswamy, N., *RFC 2405, The ESP DES-CBC Cipher Algorithm with Explicit IV*, The Internet Society, 1998.

Madsen, C. and Glenn, R., *RFC 2403, The Use of HMAC-MD5-96 within ESP and AH*, The Internet Society, 1998.

Madsen, C. and Glenn, R., *RFC 2404, The Use of HMAC-SHA-1-96 within ESP and AH*, The Internet Society, 1998.

Pereira, R. and Adams, R., *RFC 2451, The ESP CBC-Mode Cipher Algorithms*, The Internet Society, 1998.

Kelly, S. and Frankel, S., *RFC 4868, Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec*, 2007.

Huttunen, A., Swander, B., Volpe, V., DiBurro, L., Stenberg, *M., RFC 3948, UDP Encapsulation of IPsec ESP Packets*, The Internet Society, 2005.

Name ipsecah, AH – IPsec Authentication Header

Synopsis `drv/ipsecah`

Description The `ipsecah` module (AH) provides strong integrity, authentication, and partial sequence integrity (replay protection) to IP datagrams. AH protects the parts of the IP datagram that can be predicted by the sender as it will be received by the receiver. For example, the IP TTL field is not a predictable field, and is not protected by AH.

AH is inserted between the IP header and the transport header. The transport header can be TCP, UDP, ICMP, or another IP header, if tunnels are being used.

AH Device AH is implemented as a module that is auto-pushed on top of IP. The entry `/dev/ipsecah` is used for tuning AH with `ndd(1M)`.

Authentication Algorithms Current authentication algorithms supported include HMAC-MD5 and HMAC-SHA-1. Each authentication algorithm has its own key size and key format properties. You can obtain a list of authentication algorithms and their properties by using the `ipsecalgs(1M)` command. You can also use the functions described in the `getipsecalgbyname(3NSL)` man page to retrieve the properties of algorithms.

Security Considerations Without replay protection enabled, AH is vulnerable to replay attacks. AH does not protect against eavesdropping. Data protected with AH can still be seen by an adversary.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsr
Interface Stability	Committed

See Also `ipsecalgs(1M)`, `ipsecconf(1M)`, `ndd(1M)`, `attributes(5)`, `getipsecalgbyname(3NSL)`, `ip(7P)`, `ipsec(7P)`, `ipsecesp(7P)`

Kent, S. and Atkinson, R.*RFC 2402, IP Authentication Header*, The Internet Society, 1998.

Name ipsecesp, ESP – IPsec Encapsulating Security Payload

Synopsis `drv/ipsecesp`

Description The `ipsecesp` module provides confidentiality, integrity, authentication, and partial sequence integrity (replay protection) to IP datagrams. The encapsulating security payload (ESP) encapsulates its data, enabling it to protect data that follows in the datagram. For TCP packets, ESP encapsulates the TCP header and its data only. If the packet is an IP in IP datagram, ESP protects the inner IP datagram. Per-socket policy allows "self-encapsulation" so ESP can encapsulate IP options when necessary. See `ipsec(7P)`.

Unlike the authentication header (AH), ESP allows multiple varieties of datagram protection. (Using a single datagram protection form can expose vulnerabilities.) For example, only ESP can be used to provide confidentiality. But protecting confidentiality alone exposes vulnerabilities in both replay attacks and cut-and-paste attacks. Similarly, if ESP protects only integrity and does not fully protect against eavesdropping, it may provide weaker protection than AH. See `ipsecah(7P)`.

ESP Device ESP is implemented as a module that is auto-pushed on top of IP. Use the `/dev/ipsecesp` entry to tune ESP with `ndd(1M)`.

Algorithms ESPuses encryption and authentication algorithms. Authentication algorithms include HMAC-MD5 and HMAC-SHA-1. Encryption algorithms include DES, Triple-DES, Blowfish and AES. Each authentication and encryption algorithm contain key size and key format properties. You can obtain a list of authentication and encryption algorithms and their properties by using the `ipsecalgs(1M)` command. You can also use the functions described in the `getipsecalgbyname(3NSL)` man page to retrieve the properties of algorithms. Because of export laws in the United States, not all encryption algorithms are available outside of the United States.

Security Considerations ESP without authentication exposes vulnerabilities to cut-and-paste cryptographic attacks as well as eavesdropping attacks. Like AH, ESP is vulnerable to eavesdropping when used without confidentiality.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsr (32-bit)
Interface Stability	Evolving

See Also `ipsecalgs(1M)`, `ipsecconf(1M)`, `ndd(1M)`, `attributes(5)`, `getipsecalgbyname(3NSL)`, `ip(7P)`, `ipsec(7P)`, `ipsecah(7P)`

Kent, S. and Atkinson, R.*RFC 2406, IP Encapsulating Security Payload (ESP)*, The Internet Society, 1998.

Name ipw – Intel Pro. Wireless 802.11b IPW2100B driver

Description The `ipw` *802.11b* wireless NIC driver is a multi-threaded, loadable, clonable, GLDv3-based STREAMS driver supporting the Data Link Provider Interface, `dlpi(7P)`, on Intel Pro Wireless 2100B chipset-based wireless NIC's. Driver functions include controller initialization, wireless *802.11b* infrastructure network connection, WEP, frame transmit and receive and promiscuous support.

Driver Configuration The `ipw` driver performs auto-negotiation to determine the data rate and mode. Supported *802.11b* data rates are 1, 2, 5.5 and 11 Mbits/sec.

The `ipw` driver supports only BSS networks (also known as "ap" or "infrastructure" networks) and the "open" ("open-system") or "shared system" authentication. Only WEP encryption is currently supported. You perform configuration and administration tasks using the `dladm(1M)` and `wificonfig(1M)` utilities.

Files

`/dev/ipw*`	Special character device.
/kernel/drv/ipw	32-bit ELF kernel module (x86).
`/kernel/drv/amd64/ipw`	64-bit ELF kernel module (x86).

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	x86

See Also `dladm(1M)`, `wificonfig(1M)`, `attributes(5)`, `gld(7D)`, `dlpi(7P)`

ANSI/IEEE Std 802.11- Standard for Wireless LAN Technology — 1999

IEEE Std 802.11b - Standard for Wireless LAN Technology-Rev. B - 2003

Name iscsi – iSCSI software initiator driver and service

Description The `iscsi` driver is a software initiator that transports SCSI commands over TCP/IP as described in *RFC 3720*.

The initiator driver is administered through `iscsiadm(1M)`. The `iscsi` initiator service is managed by the service management facility, `smf(5)`, under the following FMRI:

```
svc:/network/iscsi/initiator:default
```

The `iscsi` initiator acts as a host adapter driver that attaches the appropriate target driver, for example, `sd(7D)` for disks, or `st(7D)` for tapes) for devices it discovers. See the *System Administration Guide: Devices and File Systems* for more information.

Once enabled, the `iscsi` initiator service ensures the right timing to start the discovery and enumeration of iSCSI devices during boot, but it doesn't guarantee the success of discovery for certain iSCSI devices. If the service is disabled, `iscsi` driver stops the discovery and enumeration of iSCSI devices and also tries to offline all existing iSCSI devices. `iscsiadm(1M)` works only when the service is enabled.

iSCSI `boot(1M)` is not affected by the status of the `iscsi` initiator service.

Files

`/kernel/drv/iscsi`	32-bit ELF kernel driver
`/kernel/drv/sparcv9/iscsi`	64-bit SPARC ELF kernel driver
`/kernel/drv/amd64/iscsi`	64-bit AMD64 ELF kernel driver
`/kernel/drv/iscsi.conf`	Driver configuration file
`/etc/iscsi/*`	`iscsi` persistent store

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWiscsir, SUNWiscsiu

See Also `iscsiadm(1M)`, `attributes(5)`, `smf(5)`, `sd(7D)`, `st(7D)`

RFC 3720 Internet Small Computer Systems Interface (iSCSI)

System Administration Guide: Devices and File Systems

Name isdnio – ISDN interfaces

Synopsis

```
#include <sun/audioio.h>
#include <sun/isdnio.h>

int ioctl(int fd, int command, /* arg */ ...);
```

Description ISDN `ioctl` commands are a subset of `ioctl(2)` commands that perform a variety of control functions on Integrated Services Digital Network (ISDN) STREAMS devices. The arguments `command` and *arg* are passed to the file designated by *fd* and are interpreted by the ISDN device driver.

fd is an open file descriptor that refers to a stream. `command` determines the control function to be performed as described in the IOCTLS section of this document. *arg* represents additional information that is needed by `command`. The type of *arg* depends upon the command, but generally it is an integer or a pointer to a command-specific data structure.

Since these ISDN commands are a subset of `ioctl` and `streamio(7I)`, they are subject to errors as described in those interface descriptions.

This set of generic ISDN `ioctl` commands is meant to control various types of ISDN STREAMS device drivers. The following paragraphs give some background on various types of ISDN hardware interfaces and data formats, and other device characteristics.

Controllers, Interfaces, and Channels

This manual page discusses operations on, and facilities provided by ISDN controllers, interfaces and channels. A controller is usually a hardware peripheral device that provides one or more ISDN interfaces and zero or more auxiliary interfaces. In this context, the term interface is synonymous with the term “port”. Each interface can provide one or more channels.

Time Division Multiplexed Serial Interfaces

ISDN BRI-TE, BRI-NT, and PRI interfaces are all examples of Time Division Multiplexed Serial Interfaces. As an example, a Basic Rate ISDN (BRI) Terminal Equipment (TE) interface provides one D-channel and two B-channels on the same set of signal wires. The BRI interface, at the S reference point, operates at a bit rate of 192,000 bits per second. The bits are encoded using a pseudoternary coding system that encodes a logic one as zero volts, and a logic zero as a positive or negative voltage. Encoding rules state that adjacent logic zeros must be encoded with opposite voltages. Violations of this rule are used to indicate framing information such that there are 4000 frames per second, each containing 48 bits. These 48 bits are divided into channels. Not including framing and synchronization bits, the frame is divided into 8 bits for the B1-channel, 1 bit for the D-channel, 8 bits for B2, 1 bit for D, 8 bits for B1, 1 bit for D, and 8 bits for B2. This results in a 64,000 bps B1-channel, a 64,000 bps B2-channel, and a 16,000 bps D-channel, all on the same serial interface.

Basic Rate ISDN

A Basic Rate ISDN (BRI) interface consists of a 16000 bit per second Delta Channel (D-channel) for signaling and X.25 packet transmission, and two 64000 bit per second Bearer Channels (B-channels) for transmission of voice or data.

The CCITT recommendations on ISDN Basic Rate interfaces, I.430, identify several "reference points" for standardization. From (Stallings89): Reference point T (terminal) corresponds to a minimal ISDN network termination at the customer's premises. It separates the network provider's equipment from the user's equipment. Reference point S (system) corresponds to the interface of individual ISDN terminals. It separates user terminal equipment from network-related communications functions. Reference point R (rate) provides a non-ISDN interface between user equipment that is not ISDN-compatible and adaptor equipment. . . . The final reference point . . . is reference point U (user). This interface describes the full-duplex data signal on the subscriber line.

Some older technology components of some ISDN networks occasionally steal the low order bit of an ISDN B-channel octet in order to transmit in-band signaling information between switches or other components of the network. Even when out-of-band signaling has been implemented in these networks, and the in-band signaling is no longer needed, the bit-robbing mechanism may still be present. This bit robbing behavior does not appreciably affect a voice call, but it will limit the usable bandwidth of a data call to 56000 bits per second instead of 64000 bits per second. These older network components only seem to exist in the United States of America, Canada and Japan. ISDN B-channel data calls that have one end point in the United States, Canada or Japan may be limited to 56000 bps usable bandwidth instead of the normal 64000 bps. Sometimes the ISDN service provider may be able to supply 56kbps for some calls and 64kbps for other calls. On an international call, the local ISDN service provider may advertise the call as 64kbps even though only 56kbps are reliably delivered because of bit-robbing in the foreign ISDN that is not reported to the local switch.

A Basic Rate Interface implements either a Terminal Equipment (TE) interface or a Network Termination (NT) interface. TE's can be ISDN telephones, a Group 4 fax, or other ISDN terminal equipment. A TE connects to an NT in order to gain access to a public or private ISDN network. A private ISDN network, such as provided by a Private Branch Exchange (PBX), usually provides access to the public network.

If multi-point configurations are allowed by an NT, it may be possible to connect up to eight TE's to a single NT interface. All of the TE's in a multipoint configuration share the same D and B-channels. Contention for B-Channels by multiple TEs is resolved by the ISDN switch (NT) through signaling protocols on the D-channel.

Contention for access to the D-channel is managed by a collision detection and priority mechanism. D-channel call control messages have higher priority than other packets. This media access function is managed at the physical layer.

A BRI-TE interface may implement a "Q-channel", the Q-channel is a slow speed, 800 bps, data path from a TE to an NT. Although the structure of the Q-channel is defined in the I.430 specification, the use of the Q-channel is for further study.

A BRI-NT interface may implement an "S-channel", the S-channel is a slow speed, 4000 bps, data path from a NT to an TE. The use of the S-channel is for further study.

Primary Rate ISDN Primary Rate ISDN (PRI) interfaces are either 1.544Mbps (T1 rate) or 2.048Mbps (E1 rate) and are typically organized as 23 B-channels and one D-Channel (23B+D) for T1 rates, and 30 B-Channels and one D-Channel (30B+D) for E1 rates. The D-channels on a PRI interface operate at 64000 bits per second. T1 rate PRI interface is the standard in the United States, Canada and Japan while E1 rate PRI interface is the standard in European countries. Some E1 rate PRI interface implementations allow access to channel zero which is used for framing.

Channel Types ISDN channels fall into several categories; D-channels, bearer channels, and management pseudo channels. Each channel has a corresponding device name somewhere under the directory `/dev/isdn/` as documented in the appropriate hardware specific manual page.

`D-channels` There is at most one D-channel per ISDN interface. The D-channel carries signaling information for the management of ISDN calls and can also carry X.25 packet data. In the case of a PRI interface, there may actually be no D-channel if Non-Facility Associated Signaling is used. D-channels carry data packets that are framed and checked for transmission errors according to the LAP-D protocol. LAP-D uses framing and error checking identical to the High Speed Data Link (HDLC) protocol.

`B-channels` BRI interfaces have two B-channels, B1 and B2. On a BRI interface, the only other type of channel is an H-channel which is a concatenation of the B1 and B2 channels. An H-channel is accessed by opening the "base" channel, B1 in this case, and using the `ISDN_SET_FORMAT` ioctl to change the configuration of the B-channel from 8-bit, 8 kHz to 16-bit, 8kHz.

On a primary rate interface, B channels are numbered from 0 to 31 in Europe and 1 to 23 in the United States, Canada and Japan.

`H-Channels` A BRI or PRI interface can offer multiple B-channels concatenated into a single, higher bandwidth channel. These concatenated B-channels are referred to as an "H-channels" on a BRI interface. The PRI interface version of an H-channel is referred to as an H*n*-channels where *n* is a number indicating how the B-channels have been aggregated into a single channel.

- A PRI interface H0 channel is 384 kbps allowing 3H0+D on a T1 rate PRI interface and 4H0+D channels on an E1 rate PRI interface.

- A T1 PRI interface H11 channel is 1536 kbps (24×64000bps). This will consume the channel normally reserved for the D-channel, so signaling must be done with Non-Facility Associated Signaling (NFAS) from another PRI interface.
- An E1 PRI interface H12 channel is 1920 kbps (30×64000bps). An H12-channel leaves room for the framing-channel as well as the D-channel.

`Auxiliary channels`

Auxiliary channels are non-ISDN hardware interfaces that are closely tied to the ISDN interfaces. An example would be a video or audio coder/decoder (codec). The existence of an auxiliary channel usually implies that one or more B-channels can be "connected" to an auxiliary interface in hardware.

`Management pseudo-channels`

A management pseudo-channel is used for the management of a controller, interface, or hardware channel. Management channels allow for out-of-band control of hardware interfaces and for out-of-band notification of status changes. There is at least one management device per hardware interface.

There are three different types of management channels implemented by ISDN hardware drivers:

- A controller management device handles all ioctls that simultaneously affect hardware channels on different interfaces. Examples include resetting a controller, mu-code (as in the Greek letter mu) downloading of a controller, or the connection of an ISDN B-channel to an auxiliary channel that represents an audio coder/decoder (codec). The latter case would be accomplished using the `ISDN_SET_CHANNEL` ioctl.
- An interface management device handles all ioctls that affect multiple channels on the same interface. Messages associated with the activation and deactivation of an interface arrive on the management device associated with the D channel of an ISDN interface.
- Auxiliary interfaces may also have management devices. See the hardware specific man pages for operations on auxiliary devices.

Trace pseudo-channels

A device driver may choose to implement a trace device for a data or management channel. Trace channels receive a special `M_PROTO` header with the original channel's original `M_PROTO` or `M_DATA` message appended to the special header. The header is described by:

```
typedef  struct {
  uint_t   seq;     /* Sequence number */
  int      type;    /* device dependent */
  struct   timeval  timestamp;
  char     _f[8];   /* filler */
} audtrace_hdr_t;
```

ISDN Channel types

The `isdn_chan_t` type enumerates the channels available on ISDN interfaces. If a particular controller implements any auxiliary channels then those auxiliary channels will be described in a controller specific manual page. The defined channels are described by the `isdn_chan_t` type as shown below:

```
/* ISDN channels */
typedef enum {
    ISDN_CHAN_NONE = 0x0,  /* No channel given */
    ISDN_CHAN_SELF,        /* The channel performing the ioctl */
    ISDN_CHAN_HOST,        /* Unix STREAMS*/
    ISDN_CHAN_CTRL_MGT,    /* Controller management */

    /* TE channel defines */

    ISDN_CHAN_TE_MGT,      /* Receives activation/deactivation */
    ISDN_CHAN_TE_D_TRACE,  /* Trace device for protocol analysis apps */
    ISDN_CHAN_TE_D,
    ISDN_CHAN_TE_B1,
    ISDN_CHAN_TE_B2,

    /* NT channel defines */

    ISDN_CHAN_NT_MGT,      /* Receives activation/deactivation */
    ISDN_CHAN_NT_D_TRACE,  /* Trace device for protocol analysis apps */
    ISDN_CHAN_NT_D,
    ISDN_CHAN_NT_B1,
    ISDN_CHAN_NT_B2,

    /* Primary rate ISDN */

    ISDN_CHAN_PRI_MGT,
    ISDN_CHAN_PRI_D,
    ISDN_CHAN_PRI_B0,  ISDN_CHAN_PRI_B1,
    ISDN_CHAN_PRI_B2,  ISDN_CHAN_PRI_B3,
    ISDN_CHAN_PRI_B4,  ISDN_CHAN_PRI_B5,
```

```
    ISDN_CHAN_PRI_B6,  ISDN_CHAN_PRI_B7,
    ISDN_CHAN_PRI_B8,  ISDN_CHAN_PRI_B9,
    ISDN_CHAN_PRI_B10, ISDN_CHAN_PRI_B11,
    ISDN_CHAN_PRI_B12, ISDN_CHAN_PRI_B13,
    ISDN_CHAN_PRI_B14, ISDN_CHAN_PRI_B15,
    ISDN_CHAN_PRI_B16, ISDN_CHAN_PRI_B17,
    ISDN_CHAN_PRI_B18, ISDN_CHAN_PRI_B19,
    ISDN_CHAN_PRI_B20, ISDN_CHAN_PRI_B21,
    ISDN_CHAN_PRI_B22, ISDN_CHAN_PRI_B23,
    ISDN_CHAN_PRI_B24, ISDN_CHAN_PRI_B25,
    ISDN_CHAN_PRI_B26, ISDN_CHAN_PRI_B27,
    ISDN_CHAN_PRI_B28, ISDN_CHAN_PRI_B29,
    ISDN_CHAN_PRI_B30, ISDN_CHAN_PRI_B31,

    /* Auxiliary channel defines */

    ISDN_CHAN_AUX0, ISDN_CHAN_AUX1, ISDN_CHAN_AUX2, ISDN_CHAN_AUX3,
    ISDN_CHAN_AUX4, ISDN_CHAN_AUX5, ISDN_CHAN_AUX6, ISDN_CHAN_AUX7
} isdn_chan_t;
```

ISDN Interface types

The `isdn_interface_t` type enumerates the interfaces available on ISDN controllers. The defined interfaces are described by the `isdn_interface_t` type as shown below:

```
/* ISDN interfaces */
typedef enum {
  ISDN_TYPE_UNKNOWN = -1,  /* Not known or applicable */
  ISDN_TYPE_SELF = 0,      /*
         * For queries, application may
         * put this value into "type" to
         * query the state of the file
         * descriptor used in an ioctl.
         */
  ISDN_TYPE_OTHER,         /* Not an ISDN interface */
  ISDN_TYPE_TE,
  ISDN_TYPE_NT,
  ISDN_TYPE_PRI,
} isdn_interface_t;
```

Activation and Deactivation of ISDN Interfaces

The management device associated with an ISDN D-channel is used to request activation, deactivation and receive information about the activation state of the interface. See the descriptions of the `ISDN_PH_ACTIVATE_REQ` and `ISDN_MPH_DEACTIVATE_REQ` ioctls. Changes in the activation state of an interface are communicated to the D-channel application through `M_PROTO` messages sent up-stream on the management device associated with the D-channel. If the D-channel protocol stack is implemented as a user process, the user process can retrieve the `M_PROTO` messages using the `getmsg(2)` system call.

These `M_PROTO` messages have the following format:

```
typedef struct isdn_message {
    unsigned int magic;           /* set to ISDN_PROTO_MAGIC */
    isdn_interface_t type;        /* Interface type */
    isdn_message_type_t message;  /* CCITT or vendor Primitive */
    unsigned int vendor[5];       /* Vendor specific content */
} isdn_message_t;
typedef  enum  isdn_message_type  {
  ISDN_VPH_VENDOR = 0,  /* Vendor specific messages */
  ISDN_PH_AI,           /* Physical: Activation Ind */
  ISDN_PH_DI,           /* Physical: Deactivation Ind */
  ISDN_MPH_AI,          /* Management: Activation Ind */
  ISDN_MPH_DI,          /* Management: Deactivation Ind */
  ISDN_MPH_EI1,         /* Management: Error 1 Indication */
  ISDN_MPH_EI2,         /* Management: Error 2 Indication */
  ISDN_MPH_II_C,        /* Management: Info Ind, connection */
  ISDN_MPH_II_D         /* Management: Info Ind, disconn. */
} isdn_message_type_t;
```

ioctls

STREAMS IOCTLS

All of the streamio(7I) `ioctl` commands may be issued for a device conforming to the the `isdnio` interface.

ISDN interfaces that allow access to audio data should implement a reasonable subset of the audio(7I) interface.

ISDN ioctls

`ISDN_PH_ACTIVATE_REQ`

Request ISDN physical layer activation. This `command` is valid for both TE and NT interfaces. *fd* must be a D-channel file descriptor. *arg* is ignored.

TE activation will occur without use of the `ISDN_PH_ACTIVATE_REQ` ioctl if the device corresponding to the TE D-channel is open, "on", and the ISDN switch is requesting activation.

`ISDN_MPH_DEACTIVATE_REQ`

fd must be an NT D-channel file descriptor. *arg* is ignored.

This `command` requests ISDN physical layer de-activation. This is not valid for TE interfaces. A TE interace may be turned off by use of the `ISDN_PARAM_POWER` command or by close(2) on the associated *fd*.

`ISDN_ACTIVATION_STATUS`

fd is the file descriptor for a D-channel, the management device associated with an ISDN interface, or the management device associated with the controller. *arg* is a pointer to an `isdn_activation_status_t` structure. Although it is possible for applications to determine the current activation state with this ioctl, a D-channel protocol stack should

instead process messages from the management pseudo channel associated with the D-channel.

```
typedef struct isdn_activation_status {
    isdn_interface_t type;
    enum isdn_activation_state activation;
} isdn_activation_status_t;
typedef enum isdn_activation_state {
    ISDN_OFF = 0,          /* Interface is powered down */
    ISDN_UNPLUGGED,        /* Power but no-physical connection */
    ISDN_DEACTIVATED_REQ,  /* Pending Deactivation, NT Only */
    ISDN_DEACTIVATED,      /* Activation is permitted */
    ISDN_ACTIVATE_REQ,     /* Attempting to activate */
    ISDN_ACTIVATED,        /* Interface is activated */
} isdn_activation_state_t;
```

The `type` field should be set to `ISDN_TYPE_SELF`. The device specific interface type will be returned in the type field.

The `isdn_activation_status_t` structure contains the interface type and the current activation state. `type` is the interface type and should be set by the caller to `ISDN_TYPE_SELF`.

`ISDN_INTERFACE_STATUS` The `ISDN_INTERFACE_STATUS` ioctl retrieves the status and statistics of an ISDN interface. The requesting channel must own the interface whose status is being requested or the ioctl will fail. *fd* is the file descriptor for an ISDN interface management device. *arg* is a pointer to a `struct isdn_interface_info`. If the `interface` field is set to `ISDN_TYPE_SELF`, it will be changed in the returned structure to reflect the proper device-specific interface of the requesting *fd*.

```
typedef struct isdn_interface_info {
  isdn_interface_t interface;
  enum isdn_activation_state    activation;
  unsigned int    ph_ai;     /* Physical: Activation Ind */
  unsigned int    ph_di;     /* Physical: Deactivation Ind */
  unsigned int    mph_ai;    /* Management: Activation Ind */
  unsigned int    mph_di;    /* Management: Deactivation Ind */
  unsigned int    mph_ei1;   /* Management: Error 1 Indication */
  unsigned int    mph_ei2;   /* Management: Error 2 Indication */
  unsigned int    mph_ii_c;  /* Management: Info Ind, connection */
  unsigned int    mph_ii_d;  /* Management: Info Ind, disconn. */
} isdn_interface_info_t;
```

`ISDN_CHANNEL_STATUS` The `ISDN_CHANNEL_STATUS` ioctl retrieves the status and statistics of an ISDN channel. The requesting channel must own the channel whose status is being requested or the ioctl will fail. *fd* is any file descriptor. *arg* is a pointer to a `struct isdn_channel_info`. If the `interface` field is set to `ISDN_CHAN_SELF`, it will be changed in the returned structure to reflect the proper device-specific channel of the requesting *fd*.

```
typedef struct isdn_channel_info {
    isdn_chan_t    channel;
    enum isdn_iostate    iostate;
    struct   isdn_io_stats {
    ulong_t  packets;   /* packets transmitted or received */
    ulong_t  octets;    /* octets transmitted or received */
    ulong_t  errors;    /* errors packets transmitted or received */
    } transmit, receive;
} isdn_channel_info_t;
```

`ISDN_PARAM_SET` *fd* is the file descriptor for a management device. *arg* is a pointer to a `struct isdn_param`. This `command` allows the setting of various ISDN physical layer parameters such as timers. This `command` uses the same arguments as the `ISDN_PARAM_GET` *command*.

`ISDN_PARAM_GET` *fd* is the file descriptor for a management device. *arg* is a pointer to a `struct isdn_param` This `command` provides for querying the value of a particular ISDN physical layer parameter.

```
typedef enum {
  ISDN_PARAM_NONE = 0,
  ISDN_PARAM_NT_T101,    /* NT Timer, 5-30 s, in milliseconds */
  ISDN_PARAM_NT_T102,    /* NT Timer, 25-100 ms, in milliseconds */
  ISDN_PARAM_TE_T103,    /* TE Timer, 5-30 s, in milliseconds */
  ISDN_PARAM_TE_T104,    /* TE Timer, 500-1000 ms, in milliseconds */
  ISDN_PARAM_MAINT,      /* Manage the TE Maintenance Channel */
  ISDN_PARAM_ASMB,       /* Modify Activation State Machine Behavior */
  ISDN_PARAM_POWER,      /* Take the interface online or offline */
  ISDN_PARAM_PAUSE,      /* Paused if == 1, else not paused == 0 */
} isdn_param_tag_t;
enum isdn_param_asmb {
    ISDN_PARAM_TE_ASMB_CCITT88, /* 1988 bluebook */
    ISDN_PARAM_TE_ASMB_CTS2,    /* Conformance Test Suite 2 */
};
typedef struct isdn_param {
    isdn_param_tag_t    tag;
    union {
```

```
   unsigned int us;          /* micro seconds */
   unsigned int ms;          /* Timer value in ms */
   unsigned int flag;        /* Boolean */
   enum isdn_param_asmb  asmb;
   enum isdn_param_maint maint;
   struct {
      isdn_chan_t channel;   /* Channel to Pause */
      int paused;            /* TRUE or FALSE */
        } pause;
   unsigned int reserved[2]; /* reserved, set to zero */
    } value;
} isdn_param_t;
```

`ISDN_PARAM_POWER` — If an implementation provides power on and off functions, then power should be on by default. If `flag` is `ISDN_PARAM_POWER_OFF` then a TE interface is forced into state F0, NT interfaces are forced into state G0. If `flag` is `ISDN_PARAM_POWER_ON` then a TE interface will immediately transition to state F3 when the TE D-channel is opened. If `flag` is one, an NT interface will transition to state G1 when the NT D-channel is opened.

Implementations that do not provide `ISDN_POWER` return failure with errno set to `ENXIO`.`ISDN_POWER` is different from `ISDN_PH_ACTIVATE_REQ` since CCITT specification requires that if a BRI-TE interface device has power, then it permits activation.

`ISDN_PARAM_NT_T101` — This parameter accesses the NT timer value T1. The CCITT recommendations specify that timer T1 has a value from 5 to 30 seconds. Other standards may differ.

`ISDN_PARAM_NT_T102` — This parameter accesses the NT timer value T2. The CCITT recommendations specify that timer T2 has a value from 25 to 100 milliseconds. Other standards may differ.

`ISDN_PARAM_TE_T103` — This parameter accesses the TE timer value T3. The CCITT recommendations specify that timer T3 has a value from 5 to 30 seconds. Other standards may differ.

`ISDN_PARAM_TE_T104` — This parameter accesses the TE timer value T4. The CTS2 specifies that timer T4 is either not used or has a value from 500 to 1000 milliseconds. Other standards may differ. CTS2 requires that timer T309 be implemented if T4 is not available.

`ISDN_PARAM_MAINT` — This parameter sets the multi-framing mode of a BRI-TE interface. For normal operation this parameter should be set

to `ISDN_PARAM_MAINT_ECHO`. Other uses of this parameter are dependent on the definition and use of the BRI interface S and Q channels.

`ISDN_PARAM_ASMB`

There are a few differences in the BRI-TE interface activation state machine standards. This parameter allows the selection of the appropriate standard. At this time, only `ISDN_PARAM_TE_ASMB_CCITT88` and `ISDN_PARAM_TE_ASMB_CTS2` are available.

`ISDN_PARAM_PAUSE`

This parameter allows a management device to pause the IO on a B-channel. `pause.channel` is set to indicate which channel is to be paused or un-paused. `pause.paused` is set to zero to un-pause and one to pause. *fd* is associated with an ISDN interface management device. *arg* is a pointer to a `struct isdn_param`.

`ISDN_SET_LOOPBACK`

fd is the file descriptor for an ISDN interface's management device. *arg* is a pointer to an `isdn_loopback_request_t` structure.

```
typedef enum {
     ISDN_LOOPBACK_LOCAL,
     ISDN_LOOPBACK_REMOTE,
} isdn_loopback_type_t;
typedef enum {
  ISDN_LOOPBACK_B1 =    0x1,
  ISDN_LOOPBACK_B2 =    0x2,
     ISDN_LOOPBACK_D  =    0x4,
     ISDN_LOOPBACK_E_ZERO = 0x8,
     ISDN_LOOPBACK_S  =    0x10,
     ISDN_LOOPBACK_Q  =    0x20,
} isdn_loopback_chan_t;
typedef struct isdn_loopback_request {
    isdn_loopback_type_t  type;
    int                   channels;
} isdn_loopback_request_t;
```

An application can receive D-channel data during D-Channel loopback but cannot transmit data. The field type is the bitwise OR of at least one of the following values:

```
ISDN_LOOPBACK_B1     (0x1)   /* loopback on B1-channel */
ISDN_LOOPBACK_B2      (0x2)   /* loopback on B2-channel */
ISDN_LOOPBACK_D       (0x4)   /* loopback on D-channel */
ISDN_LOOPBACK_E_ZERO  (0x8)   /* force E-channel to Zero if */
                               /* fd is for NT interface */
ISDN_LOOPBACK_S       (0x10)  /* loopback on S-channel */
```

```
  ISDN_LOOPBACK_Q       (0x20)  /* loopback on Q-channel */
```

`ISDN_RESET_LOOPBACK` — *arg* is a pointer to an `isdn_loopback_request_t` structure. `ISDN_RESET_LOOPBACK` turns off the selected loopback modes.

ISDN Data Format

The `isdn_format_t` type is meant to be a complete description of the various data modes and rates available on an ISDN interface. Several macros are available for setting the format fields. The `isdn_format_t` structure is shown below:

```
/* ISDN channel data format */
typedef enum {
  ISDN_MODE_NOTSPEC,     /* Not specified */
  ISDN_MODE_HDLC,        /* HDLC framing and error checking */
  ISDN_MODE_TRANSPARENT  /* Transparent mode */
} isdn_mode_t;

/* Audio encoding types (from audioio.h) */

#define AUDIO_ENCODING_NONE   (0)  /* no encoding*/
#define AUDIO_ENCODING_ULAW   (1)  /* mu-law */
#define AUDIO_ENCODING_ALAW   (2)  /* A-law */
#define AUDIO_ENCODING_LINEAR (3)  /* Linear PCM */
typedef struct isdn_format {
  isdn_mode_t    mode;
  unsigned int    sample_rate;  /* sample frames/sec*/
  unsigned int    channels;     /* # interleaved chans */
  unsigned int    precision;    /* bits per sample */
  unsigned int    encoding;     /* data encoding */
} isdn_format_t;
/*
 * These macros set the fields pointed
 * to by the macro argument (isdn_format_t*)fp in preparation
 * for the ISDN_SET_FORMAT ioctl.
 */
ISDN_SET_FORMAT_BRI_D(fp)      /* BRI D-channel */
ISDN_SET_FORMAT_PRI_D(fp)      /* PRI D-channel */
ISDN_SET_FORMAT_HDLC_B64(fp)   /* BRI B-ch @ 56kbps */
ISDN_SET_FORMAT_HDLC_B56(fp)   /* BRI B-ch @ 64kbps */
ISDN_SET_FORMAT_VOICE_ULAW(fp) /* BRI B-ch voice */
ISDN_SET_FORMAT_VOICE_ALAW(fp) /* BRI B-ch voice */
ISDN_SET_FORMAT_BRI_H(fp)      /* BRI H-channel */
```

ISDN Datapath Types

Every STREAMS stream that carries data to or from the ISDN serial interfaces is classified as a channel-stream datapath. A possible ISDN channel-stream datapath device name for a TE could be `/dev/isdn/0/te/b1`.

On some hardware implementations, it is possible to route the data from hardware channel to hardware channel completely within the chip or controller. This is classified as a channel-channel datapath. There does not need to be any open file descriptor for either channel in this configuration. Only when data enters the host and utilizes a STREAMS stream is this classified as an ISDN channel-stream datapath.

ISDN Management Stream

A management stream is a STREAMS stream that exists solely for control purposes and is not intended to carry data to or from the ISDN serial interfaces. A possible management device name for a TE could be `/dev/isdn/0/te/mgt`.

Channel Management Ioctls

The following ioctls describe operations on individual channels and the connection of multiple channels.

`ISDN_SET_FORMAT`

fd is a data channel, the management pseudo-channel associated with the data channel, or the management channel associated with the data channel's interface or controller. *arg* is a pointer to a `struct isdn_format_req`. The `ISDN_SET_FORMAT` ioctl sets the format of an ISDN channel-stream datapath. It may be issued on both an open ISDN channel-stream datapath Stream or an ISDN Management Stream. Note that an open(2) call for a channel-stream datapath will fail if an `ISDN_SET_FORMAT` has never been issued after a reset, as the mode for all channel-stream datapaths is initially biased to `ISDN_MODE_NOTSPEC`. *arg* is a pointer to an ISDN format type (`isdn_format_req_t*`).

```
typedef struct isdn_format_req {
  isdn_chan_t    channel;
  isdn_format_t format;   /* data format */
  int reserved[4];        /* future use - must be 0 */
} isdn_format_req_t;
```

If there is not an open channel-stream datapath for a requested channel, the default format of that channel will be set for a subsequent open(2).

To modify the format of an open stream, the driver will disconnect the hardware channel, flush the internal hardware queues, set the new default configuration, and finally reconnect the data path using the newly specified format. Upon taking effect, all state information will be reset to initial conditions, as if a channel was just opened. It is suggested that the user flush the interface as well as consult the hardware specific documentation to insure data integrity.

If a user desires to connect more than one B channel, such as an H-channel, the B-channel with the smallest offset should be specified, then the precision should be specified multiples of 8. For an

H-channel the precision value would be 16. The user should subsequently open the base B-channel. If any of the sequential B-channels are busy the open will fail, otherwise all of the B-channels that are to be used in conjunction will be marked as busy.

The returned failure codes and their descriptions are listed below:

```
EPERM   /* No permission for intented operation */
EINVAL   /* Invalid format request */
EIO      /* Set format attempt failed. */
```

`ISDN_SET_CHANNEL` The `ISDN_SET_CHANNEL` ioctl sets up a data connection within an ISDN controller. The `ISDN_SET_CHANNEL` `ioctl` can only be issued from an ISDN management stream to establish or modify channel-channel datapaths. The ioctl parameter *arg* is a pointer to an ISDN connection request (`isdn_conn_req_t*`). Once a data path is established, data flow is started as soon as the path endpoints become active. Upon taking effect, all state information is reset to initial conditions, as if a channel was just opened.

The `isdn_conn_req_t` structure is shown below. The five fields include the receive and transmit ISDN channels, the number of directions of the data path, as well as the data format. The reserved field must always be set to zero.

```
/* Number of directions for data flow */
typedef enum {
  ISDN_PATH_NOCHANGE = 0, /* Invalid value */
  ISDN_PATH_DISCONNECT,   /* Disconnect data path */
  ISDN_PATH_ONEWAY,       /* One way data path */
  ISDN_PATH_TWOWAY,       /* Bi-directional data path */
} isdn_path_t;
typedef struct isdn_conn_req {
  isdn_chan_t  from;
  isdn_chan_t  to;
  isdn_path_t  dir;      /* uni/bi-directional or disconnect */
  isdn_format_t format;  /* data format */
  int  reserved[4];      /* future use - must be 0 */
} isdn_conn_req_t;
```

To specify a read-only, write-only, or read-write path, or to disconnect a path, the `dir` field should be set to `ISDN_PATH_ONEWAY`, `ISDN_PATH_TWOWAY`, and `ISDN_PATH_DISCONNECT` respectively. To modify the format of a channel-channel datapath, a user must disconnect the channel and then reconnect with the desired format.

The returned failure codes and their descriptions are listed below:

```
EPERM   /* No permission for intented operation */
EBUSY   /* Connection in use */
EINVAL  /* Invalid connection request */
EIO     /* Connection attempt failed */
```

`ISDN_GET_FORMAT` The `ISDN_GET_FORMAT` ioctl gets the ISDN data format of the channel-stream datapath described by *fd*. *arg* is a pointer to an ISDN data format request type (`isdn_format_req_t*`). `ISDN_GET_FORMAT` can be issued on any channel to retrieve the format of any channel it owns. For example, if issued on the TE management channel, the format of any other te channel can be retrieved.

`ISDN_GETCONFIG` The `ISDN_GETCONFIG` ioctl is used to get the current connection status of all ISDN channels associated with a particular management stream. `ISDN_GETCONFIG` also retrieves a hardware identifier and the generic interface type. *arg* is an ISDN connection table pointer (`isdn_conn_tab_t*`). The `isdn_conn_tab_t` structure is shown below:

```
typedef struct isdn_conn_tab {
  char name[ISDN_ID_SIZE];  /* identification string */
  isdn_interface_t type;
  int maxpaths;             /* size in entries of app's array int npaths; */
                            /* number of valid entries returned by driver */
  isdn_conn_req_t *paths;   /* connection table in app's memory */
} isdn_conn_tab_t;
```

The table contains a string which is the interface's unique identification string. The second element of this table contains the ISDN transmit and receive connections and configuration for all possible data paths for each type of ISDN controller hardware. Entries that are not connected will have a value of `ISDN_NO_CHAN` in the `from` and `to` fields. The number of entries will always be `ISDN_MAX_CHANS`, and can be referenced in the hardware specific implementation documentation. An `isdn_conn_tab_t` structure is allocated on a per controller basis.

See Also `getmsg(2)`, `ioctl(2)`, `open(2)`, `poll(2)`, `read(2)`, `write(2)`, `audio(7I)`, `streamio(7I)`

ISDN, An Introduction – William Stallings, Macmillan Publishing Company. ISBN 0-02-415471-7

Name iser – iSCSI Extensions for Remote DMA driver

Description The iSER driver accelerates the iSCSI protocol by mapping the data transfer phases to Remote DMA (RDMA) operations. No iSER configuration is required for its use, but an RDMA-capable protocol (RCaP) must be configured and enabled on both target and initiator endpoints.

Currently, InfiniBand RC is the supported RCaP, and for discovery IP over IB must be configured on both the initiator and target. If Infiniband (IB) hardware is present and an Infiniband reliable-connected (RC) connection can be established then an iSER-enabled initiator uses iSER connections to iSER-enabled targets. Otherwise the connection is established using IP-based connectivity.

Files

`/kernel/drv/iser`	32–bit ELF kernel driver
`/kernel/drv/sparcv9/iser`	64-bit SPARC ELF kernel drive
`/kernel/drv/amd64/iser`	64-bit AMD64 ELF kernel driver
`/kernel/drv/iser.conf`	Driver configuration file

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWiscsidmr

See Also `iscsiadm(1M)`, `itadm(1M)`, `attributes(5)`, ibd(7D)

System Administration Guide: Devices and File Systems

RFC 3720 Internet Small Computer Systems Interface (iSCSI)

RFC 5046 iSCSI Extensions for RDM

Name isp – ISP SCSI Host Bus Adapter Driver

Synopsis

Sbus `QLGC,isp@`*sbus-slot*`,10000`

PCI `SUNW,isptwo@`*pci-slot*

Description The ISP Host Bus Adapter is a SCSA compliant nexus driver that supports the Qlogic ISP1000 SCSI and the ISP1040B SCSI chips. The ISP1000 chip works on SBus and the ISP1040B chip works on PCI bus. The ISP is an intelligent SCSI Host Bus Adapter chip that reduces the amount of CPU overhead used in a SCSI transfer.

The `isp` driver supports the standard functions provided by the SCSA interface. The driver supports tagged and untagged queuing, fast and wide SCSI, and auto request sense, but does not support linked commands. The PCI version ISP Host bus adapter based on ISP1040B also supports Fast-20 scsi devices.

Configuration The `isp` driver can be configured by defining properties in `isp.conf` which override the global SCSI settings. Supported properties are `scsi-options`, `target`<*n*>`-scsi-options`, `scsi-reset-delay`, `scsi-watchdog-tick`, `scsi-tag-age-limit`, `scsi-initiator-id`, and `scsi-selection-timeout`.

`target`<*n*>`-scsi-options` overrides the `scsi-options` property value for `target`<*n*>. <*n*> is a hex value that can vary from `0` to `f`. Refer to `scsi_hba_attach(9F)` for details.

Both the ISP1000 and ISP1040B support only certain SCSI selection timeout values. The valid values are 25, 50, 75, 100, 250, 500, 750 and 1000. These properties are in units of milliseconds.

Examples **EXAMPLE 1** SCSI Options

Create a file called `/kernel/drv/isp.conf` and add this line:

```
scsi-options=0x78;
```

This will disable tagged queuing, fast SCSI, and Wide mode for all `isp` instances. The following will disable an option for one specific ISP (refer to `driver.conf(4)`):

```
name="isp"  parent="/iommu@f,e0000000/sbus@f,e0001000"
       reg=1,0x10000,0x450
       target1-scsi-options=0x58
       scsi-options=0x178 scsi-initiator-id=6;
```

Note that the default initiator ID in OBP is 7 and that the change to ID 6 will occur at attach time. It may be preferable to change the initiator ID in OBP.

The above would set `scsi-options` for target 1 to `0x58` and for all other targets on this SCSI bus to `0x178`.

The physical pathname of the parent can be determined using the `/devices` tree or following the link of the logical device name:

EXAMPLE 1 SCSI Options *(Continued)*

```
example# ls -l /dev/rdsk/c2t0d0s0
lrwxrwxrwx   1 root   root  76 Aug 22 13:29 /dev/rdsk/c2t0d0s0 ->
../../devices/iommu@f,e0000000/sbus@f,e0001000/QLGC,isp@1,10000/sd@0,0:a,raw
```

Determine the register property values using the output of `prtconf(1M)` with the `-v` option:

```
QLGC,isp, instance #0
    ...
    Register Specifications:
        Bus Type=0x1, Address=0x10000, Size=450
```

EXAMPLE 2 ISP Properties

The `isp` driver exports properties indicating per target the negotiated transfer speed (`target<n>-sync-speed`), whether tagged queuing has been enabled (`target<n>-TQ`), and whether the wide data transfer has been negotiated (`target<n>-wide`). The `sync-speed` property value is the data transfer rate in KB/sec. The `target-TQ` and `target-wide` properties have no value. The existence of these properties indicate that tagged queuing or wide transfer has been enabled. Refer to `prtconf(1M)` (verbose option) for viewing the `isp` properties.

```
QLGC,isp, instance #2
    Driver software properties:
         name <target0-TQ> length <0> -- <no value>.
         name <target0-wide> length <0> -- <no value>.
         name <target0-sync-speed> length <4>
             value <0x000028f5>.
         name <scsi-options> length <4>
             value <0x000003f8>.
         name <scsi-watchdog-tick> length <4>
             value <0x0000000a>.
         name <scsi-tag-age-limit> length <4>
             value <0x00000008>.
         name <scsi-reset-delay> length <4>
             value <0x00000bb8>.
```

EXAMPLE 3 PCI Bus

To achieve the same setting of SCSI-options as in `instance #0` above on a PCI machine, create a file called `/kernel/drv/isp.conf` and add the following entries.

```
name="isp" parent="/pci@1f,2000/pci@1"
  unit-address="4"
  scsi-options=0x178
  target3-scsi-options=0x58 scsi-initiator-id=6;
```

EXAMPLE 3 PCI Bus *(Continued)*

The physical pathname of the parent can be determined using the `/devices` tree or following the link of the logical device name:

To set scsi-options more specifically per device type, add the following line in the `/kernel/drv/isp.conf` file:

```
device-type-scsi-options-list =
    "SEAGATE ST32550W", "seagate-scsi-options" ;
seagate-scsi-options = 0x58;
```

All device which are of this specific disk type will have `scsi-options` set to 0x58.

`scsi-options` specified per target ID has the highest precedence, followed by `scsi-options` per device type. Global (for all `isp` instances) `scsi-options` per bus has the lowest precedence.

The system needs to be rebooted before the specified `scsi-options` take effect.

EXAMPLE 4 Driver Capabilities

The target driver needs to set capabilities in the `isp` driver in order to enable some driver features. The target driver can query and modify these capabilities: `synchronous`, `tagged-qing`, `wide-xfer`, `auto-rqsense`, `qfull-retries`, `qfull-retry-interval`. All other capabilities can only be queried.

By default, `tagged-qing`, `auto-rqsense`, and `wide-xfer` capabilities are disabled, while `disconnect`, `synchronous`, and `untagged-qing` are enabled. These capabilities can only have binary values (`0` or `1`). The default values for `qfull-retries` and `qfull-retry-interval` are both `10`. The `qfull-retries` capability is a `uchar_t` (`0` to `255`) while `qfull-retry-interval` is a `ushort_t` (`0` to `65535`).

The target driver needs to enable `tagged-qing` and `wide-xfer` explicitly. The `untagged-qing` capability is always enabled and its value cannot be modified, because `isp` can queue commands even when `tagged-qing` is disabled.

Whenever there is a conflict between the value of `scsi-options` and a capability, the value set in `scsi-options` prevails. Only `whom != 0` is supported in the `scsi_ifsetcap(9F)` call.

Refer to `scsi_ifsetcap(9F)` and `scsi_ifgetcap(9F)` for details.

Files

`/kernel/drv/isp`	ELF Kernel Module
`/kernel/drv/isp.conf`	Configuration file

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC

See Also prtconf(1M), driver.conf(4), attributes(5), scsi_abort(9F), scsi_hba_attach(9F), scsi_ifgetcap(9F), scsi_reset(9F), scsi_transport(9F), scsi_device(9S), scsi_extended_sense(9S), scsi_inquiry(9S), scsi_pkt(9S)

Writing Device Drivers

ANSI Small Computer System Interface-2 (SCSI-2)

QLogic Corporation, *ISP1000 Firmware Interface Specification*

QLogic Corporation, *ISP1020 Firmware Interface Specification*

QLogic Corporation, *ISP1000 Technical Manual*

QLogic Corporation, *ISP1020a/1040a Technical Manual*

QLogic Corporation, *Differences between the ISP1020a/1040a and the ISP1020B/1040B - Application Note*

Diagnostics The messages described below may appear on the system console as well as being logged.

The first set of messages may be displayed while the `isp` driver is first trying to attach. All of these messages mean that the `isp` driver was unable to attach. These messages are preceded by "isp<number>", where "<number>" is the instance number of the ISP Host Bus Adapter.

`Device in slave-only slot, unused` — The SBus device has been placed in a slave-only slot and will not be accessible; move to non-slave-only SBus slot.

`Device is using a hilevel intr, unused` — The device was configured with an interrupt level that cannot be used with this `isp` driver. Check the device.

`Failed to alloc soft state` — Driver was unable to allocate space for the internal state structure. Driver did not attach to device; SCSI devices will be inaccessible.

`Bad soft state` — Driver requested an invalid internal state structure. Driver did not attach to device; SCSI devices will be inaccessible.

`Unable to map registers`	Driver was unable to map device registers; check for bad hardware. Driver did not attach to device; SCSI devices will be inaccessible.
`Cannot add intr`	Driver was not able to add the interrupt routine to the kernel. Driver did not attach to device; SCSI devices will be inaccessible.
`Unable to attach`	Driver was unable to attach to the hardware for some reason that may be printed. Driver did not attach to device; SCSI devices will be inaccessible.

The next set of messages can be displayed at any time. They will be printed with the full device pathname followed by the shorter form described above.

`Firmware should be < 0x<number> bytes`	Firmware size exceeded allocated space and will not download firmware. This could mean that the firmware was corrupted somehow. Check the `isp` driver.
`Firmware checksum incorrect`	Firmware has an invalid checksum and will not be downloaded.
`Chip reset timeout`	ISP chip failed to reset in the time allocated; may be bad hardware.
`Stop firmware failed`	Stopping the firmware failed; may be bad hardware.
`Load ram failed`	Unable to download new firmware into the ISP chip.
`DMA setup failed`	The DMA setup failed in the host adapter driver on a `scsi_pkt`. This will return `TRAN_BADPKT` to a SCSA target driver.
`Bad request pkt`	The ISP Firmware rejected the packet as being set up incorrectly. This will cause the `isp` driver to call the target completion routine with the reason of `CMD_TRAN_ERR` set in the `scsi_pkt`. Check the target driver for correctly setting up the packet.

`Bad request pkt header`	The ISP Firmware rejected the packet as being set up incorrectly. This will cause the `isp` driver to call the target completion routine with the reason of `CMD_TRAN_ERR` set in the `scsi_pkt`. Check the target driver for correctly setting up the packet.
`Polled command timeout on <number>.<number>`	A polled command experienced a timeout. The target device, as noted by the target lun (<number>.<number>) information, may not be responding correctly to the command, or the ISP chip may be hung. This will cause an error recovery to be initiated in the `isp` driver. This could mean a bad device or cabling.
`SCSI Cable/Connection problem`	
`Hardware/Firmware error`	The ISP chip encountered a firmware error of some kind. The problem is probably due to a faulty scsi cable or improper cable connection. This error will cause the `isp` driver to do error recovery by resetting the chip.
`Received unexpected SCSI Reset`	The ISP chip received an unexpected SCSI Reset and has initiated its own internal error recovery, which will return all the `scsi_pkt` with reason set to `CMD_RESET`.
`Fatal timeout on target <number>.<number>`	The `isp` driver found a command that had not completed in the correct amount of time; this will cause error recovery by the `isp` driver. The device that experienced the timeout was at target lun (<number>.<number>).
`Fatal error, resetting interface`	This is an indication that the `isp` driver is doing error recovery. This will cause all outstanding commands that have been transported to the `isp` driver to be completed via the `scsi_pkt` completion routine in the target driver with reason

of `CMD_RESET` and status of `STAT_BUS_RESET` set in the `scsi_pkt`.

Name iwh – Intel(R) WiFi Link 5100/5300 Driver

Description The `iwh` *802.11a/g/n* wireless NIC driver is a multi-threaded, loadable, clonable, GLDv3-based STREAMS driver supporting the Intel Shirley Peak WiFi chipset-based NIC's. Driver functions include controller initialization, wireless *802.11* infrastructure network connection, WEP and frame transmit and receive.

Configuration The `iwh` driver performs auto-negotiation to determine the data rate and mode. Supported *802.11b* data rates are 1, 2, 5.5 and 11 Mbits/sec. Supported *802.11g* data rates are 1, 2, 5.5, 11, 6, 9, 12, 18, 24, 36, 48 and 54 Mbits/sec.

Files

`/dev/iwh`	Special character device.
`/kernel/drv/iwh`	32-bit ELF kernel module (x86).
`/kernel/drv/amd64/iwh`	64-bit ELF kernel module (x86).

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWiwh
Interface Stability	Committed

See Also `dladm(1M)`, `wificonfig(1M)`, `attributes(5)`, `gld(7D)`, `dlpi(7P)`

802.11 - Wireless LAN Media Access Control and Physical Layer Specification– IEEE, 2001.

Name iwi – Intel Pro. Wireless 802.11a/b/g IPW2200B/G IPW2915A/B/G Driver

Description The `iwi` *802.11b/g* wireless NIC driver is a multi-threaded, loadable, clonable, GLDv3-based STREAMS driver supporting the Data Link Provider Interface, `dlpi(7P)`, on Intel Pro Wireless 2200BG 2915ABG chipset-based wireless NIC's. Driver functions include controller initialization, wireless *802.11b* infrastructure network connection, WEP and frame transmit and receive.

Driver Configuration The `iwi` driver performs auto-negotiation to determine the data rate and mode. Supported *802.11b* data rates are 1, 2, 5.5 and 11 Mbits/sec. Supported *802.11g* data rates are 1, 2, 5.5, 11, 6, 9, 12, 18, 24, 36, 48 and 54 Mbits/sec.

The `iwi` driver supports only BSS networks (also known as "ap" or "infrastructure" networks) and "open"(or "open-system") or "shared system" authentication. Only WEP encryption is currently supported. You perform configuration and administration tasks using the `dladm(1M)` and `wificonfig(1M)` utilities.

Files

`/dev/iwi*`	Special character device.
/kernel/drv/iwi	32-bit ELF kernel module (x86).
`/kernel/drv/amd64/iwi`	64-bit ELF kernel module (x86).

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	x86

See Also `dladm(1M)`, `wificonfig(1M)`, `attributes(5)`, `gld(7D)`, `dlpi(7P)`

ANSI/IEEE Std 802.11- Standard for Wireless LAN Technology — 1999

IEEE Std 802.11a- Standard for Wireless LAN Technology-Rev. A— 2003

IEEE Std 802.11b - Standard for Wireless LAN Technology-Rev. B — 2003

IEEE Std 802.11g- Standard for Wireless LAN Technology -Rev. G— 2003

Name iwk – Intel Pro. Wireless 802.11a/g/n 4965 driver

Description The iwk *802.11a/g/n* wireless NIC driver is a multi-threaded, loadable, clonable, GLDv3-based STREAMS driver supporting the Intel Pro Wireless 4965AGN chipset-based wireless NIC's.

Driver Configuration The `iwk` driver supports only *802.11g* BSS networks (also known as "ap" or "infrastructure" networks) and "open" (or "open-system") or "shared system" authentication. For wireless security, WEP encryption and WPA-PSK are currently supported. You perform configuration and administration tasks using the `dladm(1M)` and `wificonfig(1M)` utilities.

Files

`/dev/iwk*`	Special character device.
`/kernel/drv/iwk`	32-bit ELF kernel module (x86).
`/kernel/drv/amd64/iwk`	64-bit ELF kernel module (x86).

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWiwk
Interface stability	Committed

See Also `dladm(1M)`, `wificonfig(1M)`, `attributes(5)`, `gld(7D)`, `dlpi(7P)`

IEEE 802.11g- Wireless LAN Standard— IEEE, 2003

Name iwp – Intel WiFi Link 6000 Series Device Driver

Description The `iwp` 802.11b/g wireless NIC driver is a multi-threaded, loadable, clonable, GLDv3-based STREAMS driver which supports the Intel WiFi Link 6000 series chipset-based NICs.

Configuration The `iwp` driver performs auto-negotiation to determine the data rate and mode. Supported 802.11b data rates are 1, 2, 5.5 and 11 Mbits/sec. Supported 802.11g data rates are 1, 2, 5.5, 11, 6, 9, 12, 18, 24, 36, 48 and 54 Mbits/sec.

The `iwp` driver only supports BSS networks, (also known as `ap` or `infrastructure` networks), `open`(or `open-system`) , or `shared system` authentication.

Files

`/kernel/drv/iwp`	32-bit ELF kernel module (x86)
`/kernel/drv/amd64/iwp`	64-bit ELF kernel module (x86)
`/dev/iwp*`	Special character device

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWiwp
Interface Stability	Committed

See Also `dladm(1M)`, `wificonfig(1M)`, `attributes(5)gld(7D)`, `dlpi(7P)`

IEEE 802.11 - *Wireless LAN Media Access Control and Physical Layer Specification*, 2001

Name ixgb – SUNWixgb, 10 Gigabit Ethernet driver for Intel 82597ex controllers and Sun Ethernet PCI-X Adapter (X5544A-4) adapters.

Synopsis `/dev/ixgb`

Description The `ixgb` 10 Gigabit Ethernet driver is a multi-threaded, loadable, clonable, GLD-based STREAMS driver supporting the Data Link Provider Interface, DLPI on Intel 82597ex 10 Gigabit Ethernet controllers and Sun 10 Gigabit Ethernet PCI-X Adapter (X5544A-4) on x86 Platforms. The Intel 10G controller incorporates both MAC and PHY functions and provides 10G (fiber) Ethernet operation on the SR and LR connectors. The Sun 10 Gigabit Ethernet PCI-X Adapter (X5544A-4) is a 133 MHz PCI-X 10 Gigabit Ethernet card utilizing the Intel 82597EX PCI-X MAC controller with XFP-based 10GigE optics.

The `ixgb` driver functions include controller initialization, frame transmit and receive, promiscuous and multicast support and error recovery and reporting.

The `ixgb` driver and hardware support auto-negotiation, a protocol specified by the *IEEE 802.3ae* specification.

Configuration The following `ixgb.conf` configuration option is supported:

default_mtu
: Upper limit on the maximum MTU size the driver allows. Intel 82597EX controller allows the configuration of jumbo frames. To configure jumbo frame, use `ifconfig(1M)`. Use ifconfig with the adapter instance and the mtu argument (for example: ifconfig ixgb0 mtu 9000) to configure the adapter for the maximum allowable jumbo frame size. Allowed range is 1500 - 9000.

Application Programming Interface The cloning character-special device /dev/ixgb is used to access all Intel 10G controllers and Sun 10 Gigabit Ethernet PCI-X adapters (X5544A-4) installed within the system

The `ixgb` driver is managed by the `dladm(1M)` command line utility, which allows VLANs to be defined on top of `ixgb` instances and for `ixgb` instances to be aggregated. See `dladm(1M)` for more details.

You must send an explicit DL_ATTACH_REQ message to associate the opened stream with a particular device (PPA). The PPA ID is interpreted as an unsigned integer data type and indicates the corresponding device instance (unit) number. The driver returns an error (`DL_ERROR_ACK`) if the PPA field value does not correspond to a valid device instance number for the system. The device is initialized on first attach and de-initialized (stopped) at last detach.

The values returned by the driver in the `DL_INFO_ACK` primitive in response to the `DL_INFO_REQ` are:

- Maximum SDU is `9000` (`ETHERMTU`, as defined in `<sys/ethernet.h>`).
- Minimum SDU is `0`.
- `DLSAP` address length is `8`.

- MAC type is `DL_ETHER`.
- `SAP` length value is –2 meaning the physical address component is followed immediately by a 2 byte `SAP` component within the DLSAP address.
- Broadcast address value is Ethernet/IEEE broadcast address (`FF:FF:FF:FF:FF:FF`).

Once in the `DL_ATTACHED` state, you must send a `DL_BIND_REQ` to associate a particular Service Access Point (SAP) with the stream.

By default, the `ixgb` driver performs auto-negotiation to select the 10G link speed.

Files

`/dev/ixgb`	Special character device.
`/kernel/drv/sparcv9/ixgb`	Driver binary.
`/kernel/drv/ixgb`	32-bit kernel module. (x86 only).
`/kernel/drv/amd64/ixgb`	64-bit kernel module (x86 only).

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86

See Also

`dladm(1M)`, `ifconfig(1M)`, `attributes(5)`, `gld(7D)`, `streamio(7I)`, `dlpi(7P)`

IEEE 802.3ae 10 Gigabit Ethernet Specification — June, 2002

Sun 10 Gigabit Ethernet PCI-X Adapter (X5544A-4) Driver Installation Notes for Solaris

Writing Device Drivers

STREAMS Programming Guide

Network Interfaces Programmer's Guide

Name ixgbe – Intel 10Gb PCI Express NIC Driver

Synopsis `/dev/ixgbe*`

Description The `ixgbe` 10 Gigabit Ethernet driver is a multi-threaded, loadable, clonable, GLD-based STREAMS driver supporting the Data Link Provider Interface, `dlpi(7P)`, on Intel 10–Gigabit PCI Express Ethernet controllers.

The `ixgbe` driver functions include controller initialization, frame transmit and receive, promiscuous and multicast support, and error recovery and reporting.

The `ixgbe` driver and hardware support auto-negotiation, a protocol specified by the *IEEE 802.3ae* specification.

Application Programming Interface The cloning character-special device, `/dev/ixgbe`, is used to access all Intel 10 –Gigabit PCI Express Ethernet devices installed within the system.

The `ixgbe` driver is managed by the `dladm(1M)` command line utility, which allows VLANs to be defined on top of `ixgbe` instances and for `ixgbe` instances to be aggregated. See `dladm(1M)` for more details.

You must send an explicit DL_ATTACH_REQ message to associate the opened stream with a particular device (PPA). The PPA ID is interpreted as an unsigned integer data type and indicates the corresponding device instance (unit) number. The driver returns an error (DL_ERROR_ACK) if the PPA field value does not correspond to a valid device instance number for the system. The device is initialized on first attach and de-initialized (stopped) at last detach.

The values returned by the driver in the DL_INFO_ACK primitive in response to your DL_INFO_REQ are:

- Maximum SDU is 16366.
- Minimum SDU is 0.
- DLSAP address length is 8.
- MAC type is DL_ETHER.
- SAP (Service Access Point) length value is -2, meaning the physical address component is followed immediately by a 2-byte SAP component within the DLSAP address.
- Broadcast address value is the Ethernet/IEEE broadcast address (FF:FF:FF:FF:FF:FF).

  Once in the DL_ATTACHED state, you must send a DL_BIND_REQ to associate a particular SAP with the stream.

Configuration By default, the `ixgbe` driver performs auto-negotiation to select the link speed and mode. Link speed and mode can only be 10000 Mbps full-duplex. See the *IEEE802.3* standard for more information.

Files

`/dev/ixgbe*`	Special character device.
`/kernel/drv/ixgbe`	32–bit device driver (x86).
`/kernel/drv/amd64/ixgbe`	64–bit device driver (x86).
`/kernel/drv/sparcv9/ixgbe`	64–bit device driver (SPARC).
`/kernel/drv/ixgbe.conf`	Configuration file.

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWixgbe
Architecture	SPARC, x86
Interface Stability	Committed

See Also

`dladm(1M)`, `netstat(1M)`, `driver.conf(4)`, `attributes(5)`, `streamio(7I)`, `dlpi(7P)`

Writing Device Drivers

STREAMS Programming Guide

Network Interfaces Programmer's Guide

IEEE 802.3ae Specification, IEEE – 2002

Name jfb – XVR-1200 Graphics Accelerator device driver

Description The `jfb` module is the device driver for the Sun XVR-600 and Sun XVR-1200 Graphics Accelerators. The XVR-1200 Graphics Accelerator is a high resolution, high performance PCI graphics framebuffer providing hardware 2D and 3D acceleration. Sun XVR-600 is the single pipeline version of the Sun XVR-1200.

The `jfbdaemon` process loads the `jfb` microcode at system startup time and during the resume sequence of a suspend-resume cycle.

Files

`/dev/fbs/jfb0`	Device special file for XVR-600 and XVR-1200 high performance single screen.
`/dev/fbs/jfb0a`	Device special file for the XVR-1200 first video out.
`/dev/fbs/jfb0b`	Device special file for the XVR-1200 second video out.
`/usr/lib/jfb.ucode`	`jfb` microcode.
`/usr/sbin/jfbdaemon`	`jfb` microcode loader.

See Also `SUNWjfb_config(1M)`

Name kb – keyboard STREAMS module

Synopsis

```
#include <sys/types.h>

#include <sys/stream.h>

#include <sys/stropts.h>

#include <sys/vuid_event.h>

#include <sys/kbio.h>

#include <sys/kbd.h>

ioctl(fd, I_PUSH, "kb");
```

Description The kb STREAMS module processes byte streams generated by a keyboard attached to a CPU serial port. Definitions for altering keyboard translation and reading events from the keyboard are contained in <sys/kbio.h> and <sys/kbd.h>.

The kb STREAMS module utilizes a set of keyboard tables to recognize which keys have been typed. Each translation table is an array of 128 16-bit words (unsigned shorts). If a table entry is less than 0x100, the entry is treated as an ISO 8859/1 character. Higher values indicate special characters that invoke more complicated actions.

Keyboard Translation Mode

The keyboard can be in one of the following translation modes:

TR_NONE	Keyboard translation is turned off and up/down key codes are reported.
TR_ASCII	ISO 8859/1 codes are reported.
TR_EVENT	firm_events are reported.
TR_UNTRANS_EVENT	firm_events containing unencoded keystation codes are reported for all input events within the window system.

Keyboard Translation-Table Entries

All instances of the kb module share seven translation tables that convert raw keystation codes to event values. The tables are:

Unshifted	Used when a key is depressed and no shifts are in effect.
Shifted	Used when a key is depressed and a Shift key is held down.
Caps Lock	Used when a key is depressed and Caps Lock is in effect.
Alt Graph	Used when a key is depressed and the Alt Graph key is held down.
Num Lock	Used when a key is depressed and Num Lock is in effect.
Controlled	Used when a key is depressed and the Control key is held down. (Regardless of whether a Shift key or the Alt Graph is being held down, or whether Caps Lock or Num Lock is in effect).

Key Up Used when a key is released.

Each key on the keyboard has a `key station` code that represents a number from 0 to 127. The number is used as an index into the translation table that is currently in effect. If the corresponding entry in the translation table is a value from 0 to 255, the value is treated as an ISO 8859/1 character, and the character is the result of the translation.

If the entry in the translation table is higher than 255, it is a special entry. Special entry values are classified according to the value of the high-order bits. The high-order value for each class is defined as a constant, as shown below. When added to the constant, the value of the low-order bits distinguish between keys within each class:

SHIFTKEYS 0x100 A shift key. The value of the particular shift key is added to determine which shift mask to apply:

- CAPSLOCK 0 Caps Lock key.
- SHIFTLOCK 1 “Shift Lock” key.
- LEFTSHIFT 2 Left-hand Shift key.
- RIGHTSHIFT 3 Right-hand Shift key.
- LEFTCTRL 4 Left-hand (or only) Control key.
- RIGHTCTRL 5 Right-hand Control key.
- ALTGRAPH 9 Alt Graph key.
- ALT 10 Alternate or Alt key.
- NUMLOCK 11 Num Lock key.

BUCKYBITS 0x200 Used to toggle mode-key-up/down status without altering the value of an accompanying ISO 8859/1 character. The actual bit-position value, minus 7, is added.

- METABIT 0 The Meta key was pressed along with the key. This is the only user-accessible bucky bit. It is ORed in as the 0x80 bit; since this bit is a legitimate bit in a character, the only way to distinguish between, for example, 0xA0 as META+0x20 and 0xA0 as an 8-bit character is to watch for META key up and META key down events and keep track of whether the META key was down.
- SYSTEMBIT 1 The System key was pressed. This is a place holder to indicate which key is the system-abort key.

FUNNY 0x300 Performs various functions depending on the value of the low 4 bits:

	NOP 0x300	Does nothing.
	OOPS 0x301	Exists, but is undefined.
	HOLE 0x302	There is no key in this position on the keyboard, and the position-code should not be used.
	RESET 0x306	Keyboard reset.
	ERROR 0x307	The keyboard driver detected an internal error.
	IDLE 0x308	The keyboard is idle (no keys down).
	COMPOSE 0x309	The COMPOSE key; the next two keys should comprise a two-character COMPOSE key sequence.
	NONL 0x30A	Used only in the Num Lock table; indicates that this key is not affected by the Num Lock state, so that the translation table to use to translate this key should be the one that would have been used had Num Lock not been in effect.
	0x30B — 0x30F	Reserved for non-parameterized functions.
FA_CLASS 0x400	A floating accent or "dead key." When this key is pressed, the next key generates an event for an accented character; for example, "floating accent grave" followed by the "a" key generates an event with the ISO 8859/1 code for the "a with grave accent" character. The low-order bits indicate which accent; the codes for the individual "floating accents" are as follows:	
	FA_UMLAUT 0x400	umlaut
	FA_CFLEX 0x401	circumflex
	FA_TILDE 0x402	tilde
	FA_CEDILLA 0x403	cedilla
	FA_ACUTE 0x404	acute accent
	FA_GRAVE 0x405	grave accent
STRING 0x500	The low-order bits index a table of strings. When a key with a `STRING` entry is depressed, the characters in the null-terminated string for that key are sent, character-by-character. The maximum length is defined as:	
	KTAB_STRLEN 10	

Individual string numbers are defined as:

HOMEARROW	0x00
UPARROW	0x01
DOWNARROW	0x02
LEFTARROW	0x03
RIGHTARROW	0x04

String numbers 0x05 — 0x0F are available for custom entries.

FUNCKEYS 0x600 There are 64 keys reserved for function keys. The actual positions are usually on the left/right/top/bottom of the keyboard.

The next-to-lowest 4 bits indicate the group of function keys:

LEFTFUNC	0x600
RIGHTFUNC	0x610
TOPFUNC 0x610	0x610
BOTTOMFUNC	0x630

The low 4 bits indicate the function key number within the group:

LF(*n*)	(LEFTFUNC+(*n*)-1)
RF(*n*)	(RIGHTFUNC+(*n*)-1)
TF(*n*)	(TOPFUNC+(*n*)-1)
BF(*n*)	(BOTTOMFUNC+(*n*)-1)

PADKEYS 0x700 A "numeric keypad key." These entries should appear only in the Num Lock translation table; when Num Lock is in effect, these events will be generated by pressing keys on the right-hand keypad. The low-order bits indicate which key. The codes for the individual keys are:

PADEQUAL 0x700	"=" key
PADSLASH 0x701	"/" key
PADSTAR 0x702	"*" key
PADMINUS 0x703	"-" key
PADSEP 0x704	"," key
PAD7 0x705	"7" key
PAD8 0x706	"8" key

PAD9 0x707	“9” key
PADPLUS 0x708	“+” key
PAD4 0x709	“4” key
PAD5 0x70A	“5” key
PAD6 0x70B	“6” key
PAD1 0x70C	“1” key
PAD2 0x70D	“2” key
PAD3 0x70E	“3” key
PAD0 0x70F	“0” key
PADDOT 0x710	“.” key
PADENTER 0x711	“Enter” key

When a function key is pressed in `TR_ASCII` mode, the following escape sequence is sent:

```
ESC[0 . . .. 9z
```

where ESC is a single escape character and “0 . .. 9” indicates the decimal representation of the function-key value. For example, function key `R1` sends the sequence:

```
ESC[208z
```

because the decimal value of RF(1) is 208. In `TR_EVENT` mode, if there is a VUID event code for the function key in question, an event with that event code is generated; otherwise, individual events for the characters of the escape sequence are generated.

Keyboard Compatibility Mode

When started, the `kb` STREAMS module is in the compatibility mode. When the keyboard is in the `TR_EVENT` translation mode, ISO 8859/1 characters from the upper half of the character set (that is, characters with the eighth bit set) , are presented as events with codes in the `ISO_FIRST` range (as defined in <<`sys/vuid_event.h`>>). For backwards compatibility with older versions of the keyboard driver, the event code is `ISO_FIRST` plus the character value. When compatibility mode is turned off, ISO 8859/1 characters are presented as events with codes equal to the character code.

Description

The following `ioctl()` requests set and retrieve the current translation mode of a keyboard:

`KIOCTRANS` Pointer to an `int`. The translation mode is set to the value in the `int` pointed to by the argument.

`KIOCGTRANS` Pointer to an `int`. The current translation mode is stored in the `int` pointed to by the argument.

`ioctl()` requests for changing and retrieving entries from the keyboard translation table use the `kiockeymap` structure:

```
struct kiockeymap {
int    kio_tablemask;    /* Translation table (one of: 0, CAPSMASK,
             * SHIFTMASK, CTRLMASK, UPMASK,
             * ALTGRAPHMASK, NUMLOCKMASK)
             */
#define KIOCABORT1 –1      /* Special “mask”: abort1 keystation */
#define KIOCABORT2 –2      /* Special “mask”: abort2 keystation */
    uchar_t kio_station; /* Physical keyboard key station (0-127) */
    ushort_t kio_entry;       /* Translation table station’s entry */
    char kio_string[10]; /* Value for STRING entries-null terminated */
};
```

`KIOCSKEY` Pointer to a `kiockeymap` structure. The translation table entry referred to by the values in that structure is changed. The `kio_tablemask` request specifies which of the following translation tables contains the entry to be modified:

UPMASK 0x0080	“Key Up” translation table.
NUMLOCKMASK 0x0800	“Num Lock” translation table.
CTRLMASK 0x0030	“Controlled” translation table.
ALTGRAPHMASK 0x0200	“Alt Graph” translation table.
SHIFTMASK 0x000E	“Shifted” translation table.
CAPSMASK 0x0001	“Caps Lock” translation table.
(No shift keys pressed or locked)	“Unshifted” translation table.

The `kio_station` request specifies the keystation code for the entry to be modified. The value of `kio_entry` is stored in the entry in question. If `kio_entry` is between STRING and STRING+15, the string contained in `kio_string` is copied to the appropriate string table entry. This call may return `EINVAL` if there are invalid arguments.

Special values of `kio_tablemask` can affect the two step “break to the PROM monitor” sequence. The usual sequence is `L1-a` or `Stop-`. If `kio_tablemask` is `KIOCABORT1`, then the value of `kio_station` is set to be the first keystation in the sequence. If `kio_tablemask`, is `KIOCABORT2` then the value of `kio_station` is set to be the second keystation in the sequence. An attempt to change the “break to the PROM monitor” sequence without having superuser permission results in an `EPERM` error.

`KIOCGKEY` The argument is a pointer to a `kiockeymap` structure. The current value of the keyboard translation table entry specified by `kio_tablemask` and `kio_station` is stored in the structure pointed to by the argument. This call may return `EINVAL` if there are invalid arguments.

KIOCTYPE The argument is a pointer to an int. A code indicating the type of the keyboard is stored in the int pointed to by the argument:

KB_SUN3 Sun Type 3 keyboard

KB_SUN4 Sun Type 4 or 5 keyboard, or non-USB Sun Type 6 keyboard

KB_USB USB standard HID keyboard, including Sun Type 6 USB keyboards

KB_ASCII ASCII terminal masquerading as keyboard

KB_PC Type 101 PC keyboard

KB_DEFAULT Stored in the int pointed to by the argument if the keyboard type is unknown. In case of error, -1 is stored in the int pointed to by the argument.

KIOCLAYOUT The argument is a pointer to an int. On a Sun Type 4 keyboard, the layout code specified by the keyboard's DIP switches is stored in the int pointed to by the argument.

KIOCCMD The argument is a pointer to an int. The command specified by the value of the int pointed to by the argument is sent to the keyboard. The commands that can be sent are:

Commands to the Sun Type 3 and Sun Type 4 keyboards:

KBD_CMD_RESET Reset keyboard as if power-up.

KBD_CMD_BELL Turn on the bell.

KBD_CMD_NOBELL Turn off the bell.

KBD_CMD_CLICK Turn on the click annunciator.

KBD_CMD_NOCLICK Turn off the click annunciator.

Commands to the Sun Type 4 keyboard:

KBD_CMD_SETLED Set keyboard LEDs.

KBD_CMD_GETLAYOUT Request that keyboard indicate layout.

Inappropriate commands for particular keyboard types are ignored. Since there is no reliable way to get the state of the bell or click (because the keyboard cannot be queried and a process could do writes to the appropriate serial driver — circumventing this ioctl() request) an equivalent ioctl() to query its state is not provided.

KIOCSLED The argument is a pointer to an char. On the Sun Type 4 keyboard, the LEDs are set to the value specified in that char. The values for the four LEDs are:

LED_CAPS_LOCK "Caps Lock" light.

`LED_COMPOSE` "Compose" light.

`LED_SCROLL_LOCK` "Scroll Lock" light.

`LED_NUM_LOCK` "Num Lock" light.

On some Japanese layouts, the value for the fifth LED is:

`LED_KANA` "Kana" light.

`KIOCGLED` Pointer to a `char`. The current state of the LEDs is stored in the `char` pointed to by the argument.

`KIOCSCOMPAT` Pointer to an `int`. "Compatibility mode" is turned on if the `int` has a value of 1, and is turned off if the `int` has a value of 0.

`KIOCGCOMPAT` Pointer to an `int`. The current state of "compatibility mode" is stored in the `int` pointed to by the argument.

The following `ioctl()` request allows the default effect of the keyboard abort sequence to be changed.

`KIOCSKABORTEN` Pointer to an `int`. The keyboard abort sequence effect (typically L1-A or Stop-A on the keyboard on SPARC systems, F1–A on x86 systems, and BREAK on the serial console device) is enabled if the `int` has a value of KIOCABORTENABLE(1). If the value is KIOCABORTDISABLE(0) , the keyboard abort sequence effect is disabled. If the value is KIOCABORTALTERNATE(2), the Alternate Break sequence is in effect and is defined by the serial console drivers `zs`(7D)`se`(7D) and `asy`(7D). Any other value of the parameter for this `ioctl()` is treated as `enable`. The Alternate Break sequence is applicable to the serial console devices only.

Due to a risk of incorrect sequence interpretation, SLIP and certain other binary protocols should not be run over the serial console port when Alternate Break sequence is in effect. Although PPP is a binary protocol, it is able to avoid these sequences using the ACCM feature in *RFC 1662*. For Solaris PPP 4.0, you do this by adding the following line to the `/etc/ppp/options` file (or other configuration files used for the connection; see `pppd`(1M) for details):

```
asyncmap  0x00002000
```

SLIP has no comparable capability, and must not be used if the Alternate Break sequence is in use.

This `ioctl()` will be active and retain state even if there is no physical keyboard in the system. The default effect (`enable`) causes the operating system to suspend and enter the kernel debugger (if present) or the

system prom (on most systems with OpenBoot proms). The default effect is enabled on most systems, but may be different on server systems with key switches in the 'secure' position. On these systems, the effect is always disabled when the key switch is in the 'secure' position. This `ioctl()` returns `EPERM` if the caller is not the superuser.

These `ioctl()` requests are supported for compatibility with the system keyboard device `/dev/kbd`.

`KIOCSDIRECT` Has no effect.

`KIOCGDIRECT` Always returns 1.

The following `ioctl()` requests are used to set and get the keyboard autorepeat delay and rate.

`KIOCSRPTDELAY` This argument is a pointer to an int, which is the kb autorepeat delay, unit in millisecond.

`KIOCGRPTDELAY` This argument is a pointer to an int. The current auto repeat delay setting is stored in the integer pointed by the argument, unit in millisecond.

`KIOCSRPTRATE` This argument is a pointer to an int, which is the kb autorepeat rate, unit in millisecond.

`KIOCGRPTRATE` This argument is a pointer to an int. The current auto repeat rate setting is stored in the integer pointed by the argument, unit in millisecond.

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable

See Also

`kbd(1)`, `kmdb(1)`, `loadkeys(1)`, `pppd(1M)`, `keytables(4)`, `attributes(5)`, `zs(7D)`, `se(7D)`, `asy(7D)`, `virtualkm(7D)`, `termio(7I)`, `usbkbm(7M)`

Notes

Many keyboards released after Sun Type 4 keyboard also report themselves as Sun Type 4 keyboards.

Name kdmouse – built-in mouse device interface

Description The `kdmouse` driver supports machines with built-in PS/2 mouse interfaces. It allows applications to obtain information about the mouse's movements and the status of its buttons.

Programs are able to read directly from the device. The data returned corresponds to the byte sequences as defined in the *IBM PS/2 Technical Reference Manual*.

Files `/dev/kdmouse` device file

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86

See Also `attributes(5)`, `vuidmice(7M)`

IBM PS/2 Technical Reference Manual.

Name kfb – Sun XVR-2500 Graphics Accelerator device driver

Description The `kfb` driver is the Sun XVR-2500 graphics accelerator device driver. The Sun XVR-2500 graphics accelerator is a high resolution, high performance PCI graphics framebuffer providing hardware 2D and 3D acceleration.

The `kfbdaemon` process provides memory management for the XVR-2500 product family.

Files

dev/fbs/kfb*n*	Device special file for XVR-2500 high performance single screen.
/usr/sbin/kfbdaemon	Memory manager.

See Also `SUNWkfb_config(1M)`

Name kmdb – Kernel debugger

Description The `kmdb` driver is the mechanism used by mdb to invoke and control `kmdb`. This is *not* a public interface.

Files /dev/kmdb Kernel debugger driver.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Private

See Also `attributes(5)`

Name kstat – kernel statistics driver

Description The `kstat` driver is the mechanism used by the `kstat(3KSTAT)` library to extract kernel statistics. This is NOT a public interface.

Files `/dev/kstat` kernel statistics driver

See Also `kstat(3KSTAT)`, `kstat(9S)`

Name ksyms – kernel symbols

Synopsis `/dev/ksyms`

Description The file `/dev/ksyms` is a character special file that allows read-only access to an ELF format image containing two sections: a symbol table and a corresponding string table. The contents of the symbol table reflect the symbol state of the currently running kernel. You can determine the size of the image with the `fstat( )` system call. The recommended method for accessing the `/dev/ksyms` file is by using the ELF access library. See `elf(3ELF)` for details. If you are not familiar with ELF format, see `a.out(4)`.

`/dev/ksyms` is an executable for the processor on which you are accessing it. It contains ELF program headers which describe the text and data segment(s) in kernel memory. Since `/dev/ksyms` has no text or data, the fields specific to file attributes are initialized to `NULL`. The remaining fields describe the text or data segment(s) in kernel memory.

`Symbol table` The SYMTAB section contains the symbol table entries present in the currently running kernel. This section is ordered as defined by the ELF definition with locally-defined symbols first, followed by globally-defined symbols. Within symbol type, the symbols are ordered by kernel module load time. For example, the kernel file symbols are first, followed by the first module's symbols, and so on, ending with the symbols from the last module loaded.

The section header index (st_shndx) field of each symbol entry in the symbol table is set to SHN_ABS, because any necessary symbol relocations are performed by the kernel link editor at module load time.

`String table` The STRTAB section contains the symbol name strings that the symbol table entries reference.

See Also `kernel(1M)`, `stat(2)`, `elf(3ELF)`, `kvm_open(3KVM)`, `a.out(4)`, `mem(7D)`

Warnings The kernel is dynamically configured. It loads kernel modules when necessary. Because of this aspect of the system, the symbol information present in the running system can vary from time to time, as kernel modules are loaded and unloaded.

When you open the `/dev/ksyms` file, you have access to an ELF image which represents a snapshot of the state of the kernel symbol information at that instant in time. While the `/dev/ksyms` file remains open, kernel module autounloading is disabled, so that you are protected from the possibility of acquiring stale symbol data. Note that new modules can still be loaded, however. If kernel modules are loaded while you have the `/dev/ksyms` file open, the snapshot held by you will not be updated. In order to have access to the symbol information of the newly loaded modules, you must first close and then reopen the `/dev/ksyms` file. Be aware that the size of the `/dev/ksyms` file will have changed. You will need to use the `fstat()` function (see `stat(2)`) to determine the new size of the file.

Avoid keeping the `/dev/ksyms` file open for extended periods of time, either by using `kvm_open(3KVM)` of the default namelist file or with a direct open. There are two reasons why you should not hold `/dev/ksyms` open. First, the system's ability to dynamically configure itself is partially disabled by the locking down of loaded modules. Second, the snapshot of symbol information held by you will not reflect the symbol information of modules loaded after your initial open of `/dev/ksyms`.

Note that the `ksyms` driver is a loadable module, and that the kernel driver modules are only loaded during an open system call. Thus it is possible to run `stat(2)` on the `/dev/ksyms` file without causing the `ksyms` driver to be loaded. In this case, the file size returned is `UNKNOWN_SIZE`. A solution for this behavior is to first open the `/dev/ksyms` file, causing the `ksyms` driver to be loaded (if necessary). You can then use the file descriptor from this open in a `fstat( )` system call to get the file's size.

Notes The kernel virtual memory access library (`libkvm`) routines use `/dev/ksyms` as the default namelist file. See `kvm_open(3KVM)` for details.

Name ldterm – standard STREAMS terminal line discipline module

Synopsis

```
#include <sys/stream.h>

#include <sys/termios.h>

int ioctl(fd,I_PUSH,"ldterm");
```

Description The `ldterm` STREAMS module provides most of the `termio(7I)` terminal interface. The `vis` module does not perform the low-level device control functions specified by flags in the `c_cflag` word of the `termio/termios` structure, or by the `IGNBRK`, `IGNPAR`, `PARMRK`, or `INPCK` flags in the `c_iflag` word of the `termio/termios` structure. Those functions must be performed by the driver or by modules pushed below the `ldterm` module. The `ldterm` module performs all other `termio/termios` functions, though some may require the cooperation of the driver or modules pushed below `ldterm` and may not be performed in some cases. These include the `IXOFF` flag in the `c_iflag` word and the delays specified in the `c_oflag` word.

The `ldterm` module also handles single and multi-byte characters from various codesets including both Extended Unix Code (EUC) and non-EUC codesets.

The remainder of this section describes the processing of various STREAMS messages on the read- and write-side.

Read-side Behavior Various types of STREAMS messages are processed as follows:

`M_BREAK` Depending on the state of the `BRKINT` flag, either an interrupt signal is generated or the message is treated as if it were an `M_DATA` message containing a single ASCII NUL character when this message is received.

`M_DATA` This message is normally processed using the standard `termio` input processing. If the `ICANON` flag is set, a single input record (“line") is accumulated in an internal buffer and sent upstream when a line-terminating character is received. If the `ICANON` flag is not set, other input processing is performed and the processed data are passed upstream.

If output is to be stopped or started as a result of the arrival of characters (usually CNTRL-Q and CNTRL-S), `M_STOP` and `M_START` messages are sent downstream. If the `IXOFF` flag is set and input is to be stopped or started as a result of flow-control considerations, `M_STOPI` and `M_STARTI` messages are sent downstream.

`M_DATA` messages are sent downstream, as necessary, to perform echoing.

If a signal is to be generated, an `M_FLUSH` message with a flag byte of `FLUSHR` is placed on the read queue. If the signal is also to flush output, an `M_FLUSH` message with a flag byte of `FLUSHW` is sent downstream.

All other messages are passed upstream unchanged.

Write-side Behavior Various types of STREAMS messages are processed as follows:

`M_FLUSH` The write queue of the module is flushed of all its data messages and the message is passed downstream.

`M_IOCTL` The function of this `ioctl` is performed and the message is passed downstream in most cases. The `TCFLSH` and `TCXONC` `ioctls` can be performed entirely in the `ldterm` module, so the reply is sent upstream and the message is not passed downstream.

`M_DATA` If the `OPOST` flag is set, or both the `XCASE` and `ICANON` flags are set, output processing is performed and the processed message is passed downstream along with any `M_DELAY` messages generated. Otherwise, the message is passed downstream without change.

`M_CTL` If the size of the data buffer associated with the message is the size of `struct iocblk`, `ldterm` will perform functional negotiation to determine where the `termio(7I)` processing is to be done. If the command field of the `iocblk` structure (`ioc_cmd`) is set to `MC_NO_CANON`, the input canonical processing normally performed on `M_DATA` messages is disabled and those messages are passed upstream unmodified. (This is for the use of modules or drivers that perform their own input processing, such as a pseudo-terminal in `TIOCREMOTE` mode connected to a program that performs this processing). If the command is `MC_DO_CANON`, all input processing is enabled. If the command is `MC_PART_CANON`, then an `M_DATA` message containing a `termios` structure is expected to be attached to the original `M_CTL` message. The `ldterm` module will examine the `iflag`, `oflag`, and `lflag` fields of the `termios` structure and from that point on, will process only those flags that have not been turned ON. If none of the above commands are found, the message is ignored. In any case, the message is passed upstream.

`M_FLUSH` The read queue of the module is flushed of all its data messages and all data in the record being accumulated are also flushed. The message is passed upstream.

`M_IOCACK` The data contained within the message, which is to be returned to the process, are augmented if necessary, and the message is passed upstream.

All other messages are passed downstream unchanged.

ioctls The `ldterm` `module` processes the following `TRANSPARENT` ioctls. All others are passed downstream.

`TCGETS/TCGETA`
The message is passed downstream. If an acknowledgment is seen, the data provided by the driver and modules downstream are augmented and the acknowledgement is passed upstream.

TCSETS/TCSETSW/TCSETSF/TCSETA/TCSETAW/TCSETAF
: The parameters that control the behavior of the ldterm module are changed. If a mode change requires options at the stream head to be changed, an M_SETOPTS message is sent upstream. If the ICANON flag is turned on or off, the read mode at the stream head is changed to message-nondiscard or byte-stream mode, respectively. If the TOSTOP flag is turned on or off, the tostop mode at the stream head is turned on or off, respectively. In any case, ldterm passes the ioctl on downstream for possible additional processing.

TCFLSH
: If the argument is 0, an M_FLUSH message with a flag byte of FLUSHR is sent downstream and placed on the read queue. If the argument is 1, the write queue is flushed of all its data messages and an M_FLUSH message with a flag byte of FLUSHW is sent upstream and downstream. If the argument is 2, the write queue is flushed of all its data messages and an M_FLUSH message with a flag byte of FLUSHRW is sent downstream and placed on the read queue.

TCXONC
: If the argument is 0 and output is not already stopped, an M_STOP message is sent downstream. If the argument is 1 and output is stopped, an M_START message is sent downstream. If the argument is 2 and input is not already stopped, an M_STOPI message is sent downstream. If the argument is 3 and input is stopped, an M_STARTI message is sent downstream.

TCSBRK
: The message is passed downstream, so the driver has a chance to drain the data and then send an M_IOCACK message upstream.

EUC_WSET
: This call takes a pointer to an eucioc structure, and uses it to set the EUC line discipline's local definition for the code set widths to be used for subsequent operations. Within the stream, the line discipline may optionally notify other modules of this setting using M_CTL messages. When this call is received and the euciocstructure contains valid data, the line discipline changes into EUC handling mode once the euciocdata is completely transferred to an internal data structure.

EUC_WGET
: This call takes a pointer to an eucioc structure, and returns in it the EUC code set widths currently in use by the EUC line discipline. If the current codeset of the line discipline is not an EUC one, the result is meaningless.

See Also termios(3C), console(7D), termio(7I)

STREAMS Programming Guide

Name llc1 – Logical Link Control Protocol Class 1 Driver

Synopsis

```
#include <sys/stropts.h>

#include <sys/ethernet.h>

#include <sys/dlpi.h>

#include <sys/llc1.h>
```

Description The `llc1` driver is a multi-threaded, loadable, clonable, STREAMS multiplexing driver supporting the connectionless Data Link Provider Interface, `dlpi(7P)`, implementing IEEE 802.2 Logical Link Control Protocol Class 1 over a STREAM to a MAC level driver. Multiple MAC level interfaces installed within the system can be supported by the driver. The `llc1` driver provides basic support for the LLC1 protocol. Functions provided include frame transmit and receive, XID, and TEST, multicast support, and error recovery and reporting.

The cloning, character-special device, `/dev/llc1`, is used to access all LLC1 controllers configured under `llc1`.

The `llc1` driver is a "Style 2" Data Link Service provider. All messages of types `M_PROTO` and `M_PCPROTO` are interpreted as DLPI primitives. An explicit `DL_ATTACH_REQ` message by the user is required to associate the opened stream with a particular device (`ppa`). The `ppa` ID is interpreted as an unsigned long and indicates the corresponding device instance (unit) number. An error (`DL_ERROR_ACK`) is returned by the driver if the `ppa` field value does not correspond to a valid device instance number for this system.

The values returned by the driver in the `DL_INFO_ACK` primitive in response to the `DL_INFO_REQ` from the user are as follows:

- The maximum Service Data UNIT (SDU) is derived from the MAC layer linked below the driver. In the case of an Ethernet driver, the SDU will be 1497.
- The minimum SDU is 0.
- The MAC type is `DL_CSMACD` or `DL_TPR` as determined by the driver linked under `llc1`. If the driver reports that it is `DL_ETHER`, it will be changed to `DL_CSMACD`; otherwise the type is the same as the MAC type.
- The `sap` length value is –1, meaning the physical address component is followed immediately by a 1-octet `sap` component within the DLSAP address.
- The service mode is `DL_CLDLS`.
- The MAC type is `DL_CSMACD` or `DL_TPR` as determined by the driver linked under `llc1`. If the driver reports that it is `DL_ETHER`, it will be changed to `DL_CSMACD`; otherwise the type is the same as the MAC type.
- The `dlsap` address length is 7.
- No optional quality of service (QOS) support is included at present, so the QOS fields should be initialized to 0.
- The DLPI version is `DL_VERSION_2`.

- The provider style is DL_STYLE2.
- The broadcast address value is the broadcast address returned from the lower level driver.

Once in the DL_ATTACHED state, the user must send a DL_BIND_REQ to associate a particular Service Access Point (SAP) with the stream. The llc1 driver interprets the sap field within the DL_BIND_REQ as an IEEE 802.2 "SAP," therefore valid values for the sap field are in the [0-0xFF] range with only even values being legal.

The llc1 driver DLSAP address format consists of the 6-octet physical (e.g., Ethernet) address component followed immediately by the 1-octet sap (type) component producing a 7-octet DLSAP address. Applications should *not* hard-code to this particular implementation-specific DLSAP address format, but use information returned in the DL_INFO_ACK primitive to compose and decompose DLSAP addresses. The sap length, full DLSAP length, and sap/physical ordering are included within the DL_INFO_ACK. The physical address length can be computed by subtracting the absolute value of the sap length from the full DLSAP address length or by issuing the DL_PHYS_ADDR_REQ to obtain the current physical address associated with the stream.

Once in the DL_BOUND state, the user may transmit frames on the LAN by sending DL_UNITDATA_REQ messages to the llc1 driver. The llc1 driver will route received frames up all open and bound streams having a sap which matches the IEEE 802.2 DSAP as DL_UNITDATA_IND messages. Received frames are duplicated and routed up multiple open streams if necessary. The DLSAP address contained within the DL_UNITDATA_REQ and DL_UNITDATA_IND messages consists of both the sap (type) and physical (Ethernet) components.

In addition to the mandatory, connectionless DLPI message set, the driver additionally supports the following primitives:

The DL_ENABMULTI_REQ and DL_DISABMULTI_REQ primitives enable/disable reception of specific multicast group addresses. A set of multicast addresses may be iteratively created and modified on a per-stream basis using these primitives. These primitives are accepted by the driver in any driver state that is valid while still being attached to the ppa.

The DL_PHYS_ADDR_REQ primitive returns the 6-octet physical address currently associated (attached) to the stream in the DL_PHYS_ADDR_ACK primitive. This primitive is valid only in states following a successful DL_ATTACH_REQ.

The DL_SET_PHYS_ADDR_REQ primitive changes the 6-octet physical address currently associated (attached) to this stream. Once changed, all streams subsequently opened and attached to this device will obtain this new physical address. Once changed, the physical address will remain set until this primitive is used to change the physical address again or the system is rebooted, whichever occurs first.

The DL_XID_REQ/DL_TEST_REQ primitives provide the means for a user to issue an LLC XID or TEST request message. A response to one of these messages will be in the form of a DL_XID_CON/DL_TEST_CON message.

The DL_XID_RES/DL_TEST_RES primitives provide a way for the user to respond to the receipt of an XID or TEST message that was received as a DL_XID_IND/DL_TEST_IND message.

XID and TEST will be automatically processed by llc1 if the DL_AUTO_XID/DL_AUTO_TEST bits are set in the DL_BIND_REQ.

Files /dev/llc1 cloning, character-special device

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86

See Also attributes(5), dlpi(7P)

Name llc2 – Class II logical link control driver

Description The `llc2` logical link control driver interfaces network software (NetBIOS, SNA, OSI, and so on) running under the Solaris operating environment to a physical LAN network controlled by one of the supported communications adapters. The `llc2` driver, which appears as a STREAMS driver to the network software, resides in the kernel and is accessed by standard UNIX STREAMS functions.

This version of the `llc2` driver includes support for both connectionless and connection-oriented logical link control class II (`llc2`) operations for Ethernet, Token Ring, and FDDI adapters when accessed through the appropriate Solaris MAC layer driver. The Data Link Provider Interface (DLPI) to the `llc2` driver enables multiple and different protocol stacks, (including NetBIOS and SNA), to operate simultaneously over one or more local area networks.

To start the `llc2` driver by default, rename file `/etc/llc2/llc2_start.default` to `/etc/llc2/llc2_start`. This allows the `/etc/rc2.d/S40llc2` script to build up the configuration file for each `ppa` interface in `/etc/llc2/default/llc2.*` and start `llc2` on each interface. To verify the configuration files, manually run `/usr/lib/llc2/llc2_autoconfig`.

For more information on the `llc2` driver, see the IEEE standard *802.2 Logical Link Control*.

Obtaining Llc2 Statistics You can obtain LLC2 statistics or reset the statistics counter to zero using the `ILD_LLC2` ioctl. The `ILD_LLC2` ioctl has a number of subcommands. The following retrieve LLC2 statistics:

Name	Function
LLC2_GET_STA_STATS	Get station statistics
LLC2_GET_SAP_STATS	Get SAP statistics
LLC2_GET_CON_STATS	Get connection statistics

The structure used depends on the subcommand sent.

Llc2_GET_STA_ Stats The `LLC2_GET_STA_STATS` command retrieves statistics on a particular Physical Point of Attachment (PPA).

When sending the `LLC2_GET_STA_STATS` command, the *llc2GetStaStats* structure is used:

```
typedef struct llc2GetStaStats {
uint_t ppa;
uint_t cmd;
uchar_t clearFlag;
uchar_t state;
ushort_t numSaps;
uchar_t saps[LLC2_MAX_SAPS];
```

```
uint_t nullSapXidCmdRcvd;
uint_t nullSapXidRspSent;
uint_t nullSapTestCmdRcvd;
uint_t nullSapTestRspSent;
uint_t outOfState;
uint_t allocFail;
uint_t protocolError;
} llc2GetStaStats_t;
```

The members of the structure are:

Member	Description
cmd	LLC2_GET_STA_STATS
clearFlag	Clear counters flag. Set this to 0 to retreive statistics and to 1 to reset all counters to 0.
state	Station component state. Possible values are ?????
numSaps	Number of active SAPs in the saps array
saps	An array of active SAP values
nullSapXidCmdRcvd	Number of NULL SAP XID commands received
nullSapXidRspSent	Number of NULL SAP XID responses sent
nullSapTestCmdRcvd	Number of NULL SAP TEST commands received
nullSapTestRspSent	Number of NULL SAP TEST responses sent
outOfState	Number of invalid events received
allocFail	Number of buffer allocation failures
protocolError	Number of protocol errors

Llc2_GET_SAP_Stats The `LLC2_GET_SAP_STATS` command retrieves statistics related to a particular SAP. When sending the `LLC2_GET_SAP_STATS` command, the *llc2GetSapStats* structure is used:

```
typedef struct llc2GetSapStats {
uint_t ppa;
uint_t cmd;
uchar_t sap;
uchar_t clearFlag;
uchar_t state;
uint_t numCons;
ushort_t cons[LLC2_MAX_CONS];
uint_t xidCmdSent;
uint_t xidCmdRcvd;
uint_t xidRspSent;
uint_t xidRspRcvd;
```

```
uint_t testCmdSent;
uint_t testCmdRcvd;
uint_t testRspSent;
uint_t testRspRcvd;
uint_t uiSent;
uint_t uiRcvd;
uint_t outOfState;
uint_t allocFail;
uint_t protocolError;
} llc2GetSapStats_t;
```

The members are:

Member	Description
ppa	Physical Point of Attachment number
cmd	LLC2_GET_SAP_STATS
sap	SAP value
clearFlag	Clear counters flag. Set this to 0 to retreive statistics and to 1 to reset all counters to 0.
state	SAP component state
numCons	Number of active connections in the cons array
cons	Array of active connection indexes
xidCmdSent	Number of XID commands sent
xidCmdRcvd	Number of XID responses received
xidRspSent	Number of XID responses sent
xidRspRcvd	Number of XID responses received
testCmdSent	Number of TEST commands sent
testCmdRcvd	Number of TEST commands received
testRspSent	Number of TEST responses sent
testRspRcvd	Number of TEST responses received
uiSent	Number of UI frames sent
uiRcvd	Number of UI frames received
outOfState	Number of invalid events received
allocFail	Number of buffer allocation failures

Member	Description
protocolError	Number of protocol errors

LIc2_GET_CON_STATS The `LLC2_GET_CON_STATS` command retrieves statistics related to a particular connection component. When sending the `LLC2_GET_CON_STATS` command, the *llc2GetConStats* structure is used:

```
typedef struct llc2GetConStats {
uint_t ppa;
uint_t cmd;
uchar_t sap;
ushort_t con;
uchar_t clearFlag;
uchar_t stateOldest;
uchar_t stateOlder;
uchar_t stateOld;
uchar_t state;
ushort_t sid;
dlsap_t rem;
ushort_t flag;
uchar_t dataFlag;
uchar_t k;
uchar_t vs;
uchar_t vr;
uchar_t nrRcvd;
ushort_t retryCount;
uint_t numToBeAcked;
uint_t numToResend;
uint_t macOutSave;
uint_t macOutDump;
uchar_t timerOn;
uint_t iSent;
uint_t iRcvd;
uint_t frmrSent;
uint_t frmrRcvd;
uint_t rrSent;
uint_t rrRcvd;
uint_t rnrSent;
uint_t rnrRcvd;
uint_t rejSent;
uint_t rejRcvd;
uint_t sabmeSent;
uint_t sabmeRcvd;
uint_t uaSent;
uint_t uaRcvd;
uint_t discSent;
uint_t outOfState;
uint_t allocFail;
```

```
uint_t protocolError;
uint_t localBusy;
uint_t remoteBusy;
uint_t maxRetryFail;
uint_t ackTimerExp;
uint_t pollTimerExp;
uint_t rejTimerExp;
uint_t remBusyTimerExp;
uint_t inactTimerExp;
uint_t sendAckTimerExp;
 } llc2GetConStats_t;
```

The members of the structure are:

Member	Description
ppa	Physical Point of Attachment number
cmd	LLC2_GET_CON_STATS
sap	SAP value
con	Connection index
clearFlag	Clear counters flag. Set this to 0 to retreive statistics and to 1 to reset all counters to 0.
stateOldest, stateOlder, stateOld, state	The four previous dlpi states of the connection
sid	SAP value and connection index
dlsap_t rem	Structure containing the remote MAC address and SAP
flag	Connection component processing flag
dataFlag	DATA_FLAG
k	transmit window size
vs	Sequence number of the next I-frame to send
vr	Sequence number of the next I-frame expected
nrRcvd	Sequence number of the last I-frame acknowledged by the remote node
retryCount	Number of timer expirations
numToBeAcked	Number of outbound I-frames to be acknowledged
numToResend	Number of outbound I-frames to be re-sent
macOutSave	Number of outbound I-frames held by the MAC driver to be saved on return to LLC2

Member	Description
macOutDump	Number of outbound I-frames held by the MAC driver to be dumped on return to LLC2
timerOn	Timer activity flag
iSent	Number of I-frames sent
iRcvd	Number of I-frames received
frmrSent	Number of frame rejects sent
frmrRcvd	Number of frame rejects received
rrSent	Number of RRs sent
rrRcvd	Number of RRs received
rnrRcvd	Number of RNRs received
rejSent	Number of rejects sent
rejRcvd	Number of rejects received
sabmeSent	Number of SABMEs sent
sabmeRcvd	Number of SABMEs received
uaSent	Number of UAs sent
uaRcvd	Number of UAs received
discSent	Number of DISCs sent
outOfState	Number of invalid events received
allocFail	Number of buffer allocation failures
protocolError	Number of protocol errors
localBusy	Number of times in a local busy state
remoteBusy	Number of times in a remote busy state
maxRetryFail	Number of failures due to reaching maxRetry
ackTimerExp	Number of ack timer expirations
pollTimerExp	Number of P-timer expirations
rejTimerExp	Number of reject timer expirations
remBusyTimerExp	Number of remote busy timer expirations
inactTimerExp	Number of inactivity timer expirations
sendAckTimerExp	Number of send ack timer expirations

Member	Description

Files

`/dev/llc2`	Clone device used to access the driver `/etc/llc2/default/llc2.?` configuration files (One file per ppa interface.)

Attributes See `attributes(5)` for a description of the following attribute:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWllc

See Also `llc2_autoconfig(1)`, `llc2_config(1)`, `llc2(4)`

Name lockstat – DTrace kernel lock instrumentation provider

Description The `lockstat` driver is a DTrace dynamic tracing provider that performs dynamic instrumentation for locking primitives in the Solaris kernel.

The `lockstat` provider makes probes available that you can use to discern lock contention statistics, or to understand virtually any aspect of locking behavior inside the operating system kernel. The `lockstat(1M)` command is implemented as a DTrace consumer that uses the lockstat provider to gather raw data.

The `lockstat` driver is not a public interface and you access the instrumentation offered by this provider through DTrace. Refer to the *Solaris Dynamic Tracing Guide* for a description of the public documented interfaces available for the DTrace facility and the probes offered by the `lockstat` provider.

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWdtrp
Interface Stability	Private

See Also `dtrace(1M)`, `lockstat(1M)`, `attributes(5)`, `dtrace(7D)`

Solaris Dynamic Tracing Guide

Name lofi – Loopback file driver

Description The `lofi` file driver exports a file as a block device, enabling system utilities such as `fstyp(1M)`, `fsck(1M)` and `mount(1M)` to operate on underlying file system images (including CD-ROM and FAT floppy images) contained on the block device. Reads and writes to the block device are translated to reads and writes on the exported file. See `lofiadm(1M)` for examples.

File block device entries are contained in `/dev/lofi`. The `/dev/rlofi` file contains the character (or raw) device entries. Entries are in the form of decimal numbers and are assigned through `lofiadm(1M)`. When created, these device entries are owned by root, in group sys and have permissions of 0600. Ownership, group, and permission settings can be altered, however there may be ramifications. See `lofiadm(1M)` for more information.

`lofi` devices can be compressed. See `lofiadm(1M)` for more information.

Files exported through `lofi` can also be encrypted. See `lofiadm(1M)` for details on how to specify encryption keys.

Files `/dev/lofictl`
Master control device

`/dev/lofi/`*n*
Block device for file *n*

`/dev/rlofi/`*n*
Character device for file *n*

`/kernel/drv/lofi`
32-bit driver

`/kernel/drv/sparcv9/lofi`
64-bit driver (SPARC)

`/kernel/drv/amd64/lofi`
64-bit driver (x86)

`/kernel/drv/lofi.conf`
Driver configuration file. (Do not alter).

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86
Availability	SUNWcsr, SUNWcarx.u
Interface Stability	Committed

See Also fstyp(1M), fsck(1M), mount(1M), lofiadm(1M), newfs(1M), attributes(5), lofs(7FS)

Notes Just as you would not directly access a disk device that has mounted file systems, you should not access a file associated with a block device except through the `lofi` file driver.

For compatibility purposes, a raw device is also exported with the block device. (For example, newfs(1M)).

The `lofi` driver isn't available in a zone and will not work inside a zone.

Name lofs – loopback virtual file system

Synopsis

```
#include <sys/param.h>
#include <sys/mount.h>

int mount (const char* dir, const char* virtual, int mflag, lofs, NULL, 0);
```

Description The loopback file system device allows new, virtual file systems to be created, which provide access to existing files using alternate pathnames. Once the virtual file system is created, other file systems can be mounted within it, without affecting the original file system. However, file systems which are subsequently mounted onto the original file system *are* visible to the virtual file system, unless or until the corresponding mount point in the virtual file system is covered by a file system mounted there.

virtual is the mount point for the virtual file system. *dir* is the pathname of the existing file system. *mflag* specifies the mount options; the `MS_DATA` bit in *mflag* must be set. If the `MS_RDONLY` bit in *mflag* is not set, accesses to the loop back file system are the same as for the underlying file system. Otherwise, all accesses in the loopback file system will be read-only. All other `mount(2)` options are inherited from the underlying file systems.

A loopback mount of '/' onto `/tmp/newroot` allows the entire file system hierarchy to appear as if it were duplicated under `/tmp/newroot`, including any file systems mounted from remote NFS servers. All files would then be accessible either from a pathname relative to '/' or from a pathname relative to `/tmp/newroot` until such time as a file system is mounted in `/tmp/newroot`, or any of its subdirectories.

Loopback mounts of '/' can be performed in conjunction with the `chroot(2)` system call, to provide a complete virtual file system to a process or family of processes.

Recursive traversal of loopback mount points is not allowed. After the loopback mount of `/tmp/newroot`, the file `/tmp/newroot/tmp/newroot` does not contain yet another file system hierarchy; rather, it appears just as `/tmp/newroot` did before the loopback mount was performed (for example, as an empty directory).

Examples `lofs` file systems are mounted using:

```
mount -F lofs /tmp /mnt
```

See Also `lofiadm(1M)`, `mount(1M)`, `chroot(2)`, `mount(2)`, `sysfs(2)`, `vfstab(4)`, `lofi(7D)`

Notes All access to entries in `lofs` mounted file systems map to their underlying file system. If a mount point is made available in multiple locations via `lofs` and is busy in any of those locations, an attempt to mount a file system at that mount point fails unless the overlay flag is specified. See `mount(1M)`. Examples of a mount point being busy within a `lofs` mount include having a file system mounted on it or it being a processes' current working directory.

Warnings Because of the potential for confusing users and applications, you should use loopback mounts with care. A loopback mount entry in `/etc/vfstab` must be placed after the mount points of both directories it depends on. This is most easily accomplished by making the loopback mount entry the last in `/etc/vfstab`.

Name log – interface to STREAMS error logging and event tracing

Synopsis

```
#include <sys/strlog.h>

#include <sys/log.h>
```

Description `log` is a STREAMS software device driver that provides an interface for console logging and for the STREAMS error logging and event tracing processes (see `strerr(1M)`, and `strace(1M)`). `log` presents two separate interfaces: a function call interface in the kernel through which STREAMS drivers and modules submit `log` messages; and a set of `ioctl(2)` requests and STREAMS messages for interaction with a user level console logger, an error logger, a trace logger, or processes that need to submit their own `log` messages.

Kernel Interface `log` messages are generated within the kernel by calls to the function `strlog()`:

```
strlog(short mid,
    short sid,
    char level,
    ushort_t flags,
    char *fmt,
    unsigned arg1 . . .
);
```

Required definitions are contained in `<sys/strlog.h>`, `<sys/log.h>`, and `<sys/syslog.h>`. *mid* is the STREAMS module id number for the module or driver submitting the `log` message. *sid* is an internal sub-id number usually used to identify a particular minor device of a driver. *level* is a tracing level that allows for selective screening out of low priority messages from the tracer. *flags* are any combination of `SL_ERROR` (the message is for the error logger), `SL_TRACE` (the message is for the tracer), `SL_CONSOLE` (the message is for the console logger), `SL_FATAL` (advisory notification of a fatal error), and `SL_NOTIFY` (request that a copy of the message be mailed to the system administrator). *fmt* is a `printf(3C)` style format string, except that `%s`, `%e`, `%E`, `%g`, and `%G` conversion specifications are not handled. Up to `NLOGARGS` (in this release, three) numeric or character arguments can be provided.

User Interface `log` is implemented as a cloneable device, it clones itself without intervention from the system clone device. Each open of `/dev/log` obtains a separate stream to `log`. In order to receive `log` messages, a process must first notify `log` whether it is an error logger, trace logger, or console logger using a STREAMS `I_STR` `ioctl` call (see below). For the console logger, the `I_STR` `ioctl` has an `ic_cmd` field of `I_CONSLOG`, with no accompanying data. For the error logger, the `I_STR` `ioctl` has an `ic_cmd` field of `I_ERRLOG`, with no accompanying data. For the trace logger, the `ioctl` has an `ic_cmd` field of `I_TRCLOG`, and must be accompanied by a data buffer containing an array of one or more struct `trace_ids` elements.

```
struct trace_ids {
    short ti_mid;
    short ti_sid;
    char  ti_level;
};
```

Each `trace_ids` structure specifies a *mid*, *sid*, and *level* from which messages will be accepted. `strlog(9F)` will accept messages whose *mid* and *sid* exactly match those in the `trace_ids` structure, and whose level is less than or equal to the level given in the `trace_ids` structure. A value of −1 in any of the fields of the `trace_ids` structure indicates that any value is accepted for that field.

Once the logger process has identified itself using the `ioctl` call, `log` will begin sending up messages subject to the restrictions noted above. These messages are obtained using the `getmsg(2)` function. The control part of this message contains a `log_ctl` structure, which specifies the *mid*, *sid*, *level*, *flags*, time in ticks since boot that the message was submitted, the corresponding time in seconds since Jan. 1, 1970, a sequence number, and a priority. The time in seconds since 1970 is provided so that the date and time of the message can be easily computed, and the time in ticks since boot is provided so that the relative timing of `log` messages can be determined.

```
struct log_ctl {
    short mid;
    short sid;
    char  level;     /* level of message for tracing */
    short flags;     /* message disposition */
#if defined(_LP64)  ||  defined(_I32LPx)
    clock32_t ltime; /* time in machine ticks since boot */
    time32_t ttime;  /* time in seconds since 1970 */
#else
    clock_t ltime;
    time_t  ttime;
#endif
    int  seq_no;     /* sequence number */
    int  pri;        /* priority = (facility|level) */
};
```

The priority consists of a priority code and a facility code, found in `<sys/syslog.h>`. If `SL_CONSOLE` is set in *flags*, the priority code is set as follows: If `SL_WARN` is set, the priority code is set to `LOG_WARNING`; If `SL_FATAL` is set, the priority code is set to `LOG_CRIT`; If `SL_ERROR` is set, the priority code is set to `LOG_ERR`; If `SL_NOTE` is set, the priority code is set to `LOG_NOTICE`; If `SL_TRACE` is set, the priority code is set to `LOG_DEBUG`; If only `SL_CONSOLE` is set, the priority code is set to `LOG_INFO`. Messages originating from the kernel have the facility code set to `LOG_KERN`. Most messages originating from user processes will have the facility code set to `LOG_USER`.

Different sequence numbers are maintained for the error and trace logging streams, and are provided so that gaps in the sequence of messages can be determined (during times of high message traffic some messages may not be delivered by the logger to avoid hogging system resources). The data part of the message contains the unexpanded text of the format string (null terminated), followed by `NLOGARGS` words for the arguments to the format string, aligned on the first word boundary following the format string.

A process may also send a message of the same structure to `log`, even if it is not an error or trace logger. The only fields of the `log_ctl` structure in the control part of the message that are accepted are the *level*, *flags*, and *pri* fields; all other fields are filled in by `log` before being forwarded to the appropriate logger. The data portion must contain a null terminated format string, and any arguments (up to `NLOGARGS`) must be packed, 32-bits each, on the next 32-bit boundary following the end of the format string.

`ENXIO` is returned for `I_TRCLOG` ioctls without any `trace_ids` structures, or for any unrecognized `ioctl` calls. The driver silently ignores incorrectly formatted `log` messages sent to the driver by a user process (no error results).

Processes that wish to write a message to the console logger may direct their output to `/dev/conslog`, using either `write(2)` or `putmsg(2)`.

Driver Configuration

The following driver configuration properties may be defined in the `log.conf` file.

msgid=1 If `msgid=1`, each message will be preceded by a message ID as described in `syslogd(1M)`.

msgid=0 If `msgid=0`, message IDs will not be generated. This property is unstable and may be removed in a future release.

Examples

EXAMPLE 1 `I_ERRLOG` registration.

```
struct strioctl ioc;
ioc.ic_cmd = I_ERRLOG;
ioc.ic_timout = 0;        /* default timeout (15 secs.) */
ioc.ic_len = 0;
ioc.ic_dp = NULL;
ioctl(log, I_STR, &ioc);
```

EXAMPLE 2 `I_TRCLOG` registration.

```
struct trace_ids tid[2];
tid[0].ti_mid = 2;
tid[0].ti_sid = 0;
tid[0].ti_level = 1;
tid[1].ti_mid = 1002;
tid[1].ti_sid = -1;              /* any sub-id will be allowed */
tid[1].ti_level = -1;            /* any level will be allowed */
ioc.ic_cmd = I_TRCLOG;
ioc.ic_timout = 0;
ioc.ic_len = 2 * sizeof(struct trace_ids);
ioc.ic_dp = (char *)tid;
ioctl(log, I_STR, &ioc);
```

Example of submitting a `log` message (no arguments):

EXAMPLE 2 I_TRCLOG registration. *(Continued)*

```
struct strbuf ctl, dat;
struct log_ctl lc;
char *message = "Don't forget to pick up some milk
                on the way home";
ctl.len = ctl.maxlen = sizeof(lc);
ctl.buf = (char *)&lc;
dat.len = dat.maxlen = strlen(message);
dat.buf = message;
lc.level = 0;
lc.flags = SL_ERROR|SL_NOTIFY;
putmsg(log, &ctl, &dat, 0);
```

Files

`/dev/log`	Log driver.
`/dev/conslog`	Write only instance of the log driver, for console logging.
`/kernel/drv/log.conf`	Log configuration file.

See Also `strace`(1M), `strerr`(1M), `Intro`(3), `getmsg`(2), `ioctl`(2), `putmsg`(2), `write`(2), `printf`(3C), `strlog`(9F)

STREAMS Programming Guide

Name lp – driver for parallel port

Synopsis

```
include <sys/bpp_io.h>
fd = open("/dev/lpn", flags);
```

Description The `lp` driver provides the interface to the parallel ports used by printers for x86 based systems. The `lp` driver is implemented as a STREAMS device.

ioctls

`BPPIOC_TESTIO` Test transfer readiness. This command checks to see if a read or write transfer would succeed based on pin status. If a transfer would succeed, `0` is returned. If a transfer would fail, `−1` is returned, and `errno` is set to `EIO`. The error status can be retrieved using the `BPPIOC_GETERR` `ioctl()` call.

`BPPIOC_GETERR` Get last error status. The argument is a pointer to a `struct bpp_error_status`. See below for a description of the elements of this structure. This structure indicates the status of all the appropriate status bits at the time of the most recent error condition during a `read(2)` or `write(2)` call, or the status of the bits at the most recent `BPPIOC_TESTIO` `ioctl(2)` call. The application can check transfer readiness without attempting another transfer using the `BPPIOC_TESTIO` `ioctl()`.

Error Pins Structure This structure and symbols are defined in the include file `<sys/bpp_io.h>`:

```
struct bpp_error_status {
       char  timeout_occurred;   /* Not use */
       char  bus_error;          /* Not use */
       uchar_t pin_status;       /* Status of pins which could cause an error */
};

/* Values for pin_status field */
#define BPP_ERR_ERR 0x01   /* Error pin active */
#define BPP_SLCT_ERR 0x02  /* Select pin active */
#define BPP_PE_ERR 0x04    /* Paper empty pin active */
```

Note: Other pin statuses are defined in `<sys/bpp_io.h>`, but `BPP_ERR_ERR`, `BPP_SLCT_ERR` and `BPP_PE_ERR` are the only ones valid for the x86 `lp` driver.

Errors

`EIO` A `BPPIOC_TESTIO` `ioctl()` call is attempted while a condition exists that would prevent a transfer (such as a peripheral error).

`EINVAL` An `ioctl()` is attempted with an invalid value in the command argument.

Files

`/platform/i86pc/kernel/drv/lp.conf` configuration file for `lp` driver

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86

See Also sysbus(4), attributes(5), streamio(7I)

Notes A read operation on a bi-directional parallel port is not supported.

Name lsimega – SCSI HBA driver for LSI MegaRAID 320-2x SCSI RAID Controller

Description The `lsimega` SCSI RAID host bus adapter driver is a SCSA-compliant nexus driver that supports LSI MegaRAID 320-2x SCSI RAID devices. The driver also supports LSI MegaRAID 320-1, 320-2e, Dell PERC 4e/Si, 4e/Di, 4e/DC RAID controllers.

The `lsimega` driver supports standard functions provided by the SCSA interface, including RAID disk I/O and passthrough I/O to support SCSI tape and CDROM.

Driver Configuration The `lsimega.conf` file contains no user configurable parameters. Please configure your hardware through BIOS.

The LSI MegaRAID 320-2x device can support up to 40 logic drivers. Note that BIOS numbers the logic drives as 1 through 40, however in Solaris these drives are numbered from 0 to 39.

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	x86

Files

`/kernel/drv/lsimega`	32-bit ELF kernel module.
`/kernel/drv/amd64/lsimega`	64–bit kernel module (x86 only).
`/kernel/drv/lsimega.conf`	Driver configuration file (contains no user-configurable options).

See Also `prtconf(1M)`, `attributes(5)`, `amr(7D)`, `scsi_hba_attach_setup(9F)`, `scsi_sync_pkt(9F)`, `scsi_transport(9F)`, `scsi_inquiry(9S)`, `scsi_device(9S)`, `scsi_pkt(9S)`

Small Computer System Interface-2 (SCSI-2)

Name lx_systrace – DTrace Linux system call tracing provider

Description The `lx_systrace` driver implements the DTrace `lxsyscall` dynamic tracing provider. The lxsyscall provider performs dynamic instrumentation to offer probes that fire whenever a thread enters or returns from a Linux system call entry point.

The `lx_systrace` driver is not a public interface and you access the instrumentation offered by this provider through DTrace. Refer to the *Solaris Dynamic Tracing Guide* for a description of the public documented interfaces available for the DTrace facility and the probes offered by the `lxsyscall` provider.

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWlxr
Interface Stability	Private

See Also `dtrace(1M)`, `attributes(5)`, `lx(5)`, `dtrace(7D)`

Solaris Dynamic Tracing Guide

Name m64 – PGX, PGX24, and PGX64 frame buffers device driver

Description The `m64` driver is the Sun PGX graphics accelerator device driver.

Files `/dev/fbs/m64\fIn` Device special file

See Also `m64config(1M)`

Name marvell88sx – Marvell 88SX SATA controller driver

Synopsis

```
sata@unit-address
```

Description The `marvell88sx` driver is a SATA framework-compliant HBA driver that supports the Marvell 88SX5081, 88SX5080, 88SX5040, 88SX5041, 88SX6081, and 88SX6041 controllers.

The 88SX5081, 88SX5080, 88SX5040 and 88SX5041 Marvell controllers are fully compliant with the Serial ATA 1.0 specification and support the SATA device hot-swap compliant 1.5 Gbps speed feature.

The 88SX6081 and 88SX6041 Marvell controllers are fully-compliant with the SATA II Phase 1.0 specification (the extension to the SATA 1.0 specification) and support SATA II native command queuing and backwards compatibility with SATA I 1.5 Gpbs speed and devices. In addition, the 88SX6081 device supports the SATA II Phase 1.0 specification features, including SATA II 3.0 Gbps speed, SATA II Port Multiplier functionality and SATA II Port Selector. Currently the driver does not support native command queuing, port multiplier or port selector functionality.

Configuration There are no tunable parameters in the `marvell88sx.conf` file.

Files

`/kernel/drv/marvell88sx`	32–bit ELF 86 kernel module.
`/kernel/drv/amd64/marvell88sx`	64–bit ELF kernel module.
`/kernel/drv/marvell88sx.conf`	Driver configuration file.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWmv88sx

See Also `cfgadm(1M)`, `prtconf(1M)`, `cfgadm_sata(1M)`, `attributes(5)`, `nv_sata(7D)`, `sata(7D)`

Diagnostics In addition to being logged, the following messages may appear on the system console:

```
marvell88sx<n>:PCI error address 0x<high addr>:<low addr>
PCI command 0x<command>DAC [true|false]  attribute
0x<attribute><pci error type>.
```

The *n*th instance of a marvell88sx reports a PCI bus status message. (A hardware issue needs attention). The PCI bus address, PCI command (whether or not it was a dual address command), the PCI-X attribute bit, and the error type are displayed.

```
marvell88sx<n>: port <port #>: error in PIO command 0<cmd>x:
status 0x<status>.
```

The port number on the *n*th marvell88sx controller received an error while performing a programmed I/O command <cmd> with status <status>.

```
marvell88sx<n>: error on port<port#>: One or more of the following:

ATA UDMA data parity error
ATA UDMA PRD parity error
device error
device disconnected
device connected
SError interrupt
reserved bit 6
EDMA self disabled
BIST FIS or asynchronous notification
command request queue parity error
command response queue parity error
internal memory parity error
I/O ready time-out
link control receive error - crc
link control receive error - fifo
link control receive error - reset
link control receive error - state
link data receive error - crc
link data receive error - fifo
link data receive error - reset
link data receive error - state
link control transmit error - crc
link control transmit error - fifo
link control transmit error - reset
link control transmit error - DMAT
link control transmit error - collision
link data transmit error -crc
link data transmit error - fifo
link data transmit error - reset
link data transmit error - DMAT
link data transmit error - collision
transport protocol error
```

The port number on the *n*th marvell88sx controller received one or more error conditions as listed.

```
marvell88sx<n>: device on port <port #> still busy.
```

The port number on the *n*th marvell88sx remains busy. (Indicates a hardware problem). Check the disk and the controller.

```
marvell88sx<n>: pci_config_setup failed.
```

Could not access PCI configuration space for the *n*th marvell88sx controller.

```
marvell88sx<n>:failed to get device id.
```

The device-id property for the *n*th marvell88sx controller cannot be read.

```
marvell88sx<n>: Unrecognized device - device id 0x<device id>
assuming <n> ports.
```

The device id associated with the *n*th marvell88sx controller is not supported and the number of ports could not be determined. *n* ports are being assumed.

```
marvell88sx<n>:Unrecognized device - device idOx<device id>.
```

The device id associated with the *n*th marvell88sx controller is not supported.

```
marvell88sx<n>: Could not attach. Could not allocate softstate.
```

A call to ddi_soft_state_zalloc() failed for the *n*th marvell88sx controller. The system may be low on resources. The driver failed to attach.

```
 marvell88sx<n>: Could not attach, unknown device model.
```

The *n*th marvell88sx controller is unsupported hardware. The driver failed to attach.

```
marvell88sx<n>: Could not attach, unsupported chip step-
 ping or unable to get the chip stepping.
```

The *n*th marvell88sx controller is not supported due to a known bad chip stepping or a stepping of an unknown model.

```
 marvell88sx<n>: ddi_intr_get_supported_types failed.
```

The driver failed to attach.

```
marvell88sx<n>: power management component not created.
```

Power management is not supported.

```
mavell88sx<n>: unable to attach to sata framework.
```

The driver failed to attach.

```
marvell88sx<n>: unable to detach from sata framework.
```

The driver failed to detach.

```
marvell88sx<n>: Could not attach, failed interrupt registration.
```

The driver failed to attach.

```
marvell88sx<n>: Cannot get number interrupts, rc
```

The number of interrupts for the *n*th marvell88sx device could not be determined.

```
 marvell88sx<n>: 0 is not a valid number of interrupts.
```

The number of interrupts for the *n*th marvell88sx device was returned as 0.

```
marvell88sx<n>: Failed to get the number of available interrupts.
```

The number of available interrupts for the *n*th marvell88sx controller could not be determined.

```
marvell88sx<n>: Number of available interrupts is 0.
```

No interrupts were available for the *n*th marvell88sx device.

```
marvell88sx<n>: could not allocate interrupts.
```

The interrupts for the *n*th marvell88sx device could not be allocated.

```
marvell88sx<n>: could not get interrupt priority.
```

The interrupt priority for the *n*th marvell88sx device could not be determined.

```
marvell88sx<n>: Could not add interrupt handler.
```

An interrupt service routine could not be added to the system for the *n*th marvell88sx device.

```
marvell88sx<n>:polled read/write request never completed- port <num>.
```

A polled read or write did not complete in a reasonable amount of time. If this problem persists, there may be a hardware problem with (a) the controller, (b) the controller port, (c) the disk attached to controller port or (d) the cabling.

```
marvell88sx<n>: EDMA never disabled.
```

Could not disable EDMA. (Indicates a hardware problem).

```
marvell88sx<n>: Could not attach.
```

The *n*th marvell8ssx device could not attach. This message is usually preceded by another warning indicating why the attach failed.

Name mc-opl – memory controller driver for the SPARC Enterprise Server family

Description The `mc-opl` driver is the memory controller driver for the SPARC Enterprise Server family. This driver manages the hardware memory-scrubbing operations.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcakr
Interface Stability	Private

See Also `attributes(5)`

Name md – user configurable pseudo device driver

Description `md` is a user configurable pseudo device driver that provides disk concatenation, striping, mirroring, RAID5 metadevices, trans metadevices, and hot spare utilities. Trans devices are no longer supported and have been replaced by UFS logging. See `mount_ufs(1M)`.

The block devices access the disk using the system's normal buffering mechanism and are read and written without regard to physical disk records. There is also a "raw" device which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call usually results in one I/O operation; raw I/O is therefore considerably more efficient when many bytes are transmitted. The names of the block devices are found in `/dev/md/dsk`; the names of the raw devices are found in `/dev/md/rdsk`. Metadevices have the appearance of whole disks; there are no slices (partitions).

I/O requests (such as `lseek(2)`) to the metadevices must have an offset that is a multiple of 512 bytes (DEV_BSIZE), or the driver returns an EINVAL error. If the transfer length is not a multiple of 512 bytes, the tranfer count is rounded up by the driver.

The `md` pseudo device drivers support all disk devices on all Solaris 2.4 or later Solaris systems.

ioctls This section provides a list of the ioctls supported by the metadisk driver.

The following ioctls are valid when issued to the raw metadevice, such as `/dev/md/rdsk/d0`. See `dkio(7I)` for additional information.

`DKIOCGGEOM` This ioctl is used to get the disk geometry. The metadisk driver fills in the `dkg_nhead`, `dkg_nsect`, `dkg_rpm`, `dkg_write_reinstruct` and `dkg_read_reinstruct` from the first component of the metadevice (at `metainit` time). `dkg_ncyl` is calculated using the size of the metadevice (reported by `metastat`) divided by (dkg_nhead * dkg_nsect). The total size is always a multiple of (dkg_nhead * dkg_nsect). If the first component of a metadevice *does not* start on cylinder number 0, then the dkg_ncyl is increased by one cylinder; because DKIOCGVTOC reports the metadevice as starting on cylinder 1. The side effect here is that it looks like cylinder 0 is not being used, but all the arithmetic works out correctly. If the metadevice is not set up, then ENXIO is returned.

`DKIOCINFO` When issued to the administrative device or metadevice, this ioctl sets `dki_unit` to the unit number of the metadevice, `dki_ctype` to a value of DKC_MD, and `dki_partition` to 0, because there are no slices.

`DKIOCGVTOC` This ioctl returns the current vtoc. If one has not been written, then a default vtoc is returned. `v_nparts` is always 1. `v_part[0].p_start` is 0 if the first component of the metadevice starts on cylinder 0. Otherwise, the `p_start` field is the starting sector of cylinder 1. `v_part[0].p_size` is the same as the total size reported by `metastat`.

`DKIOCSVTOC` This ioctl stores the vtoc in the metadevice state database so it is persistent across reboots.

Diagnostics

Notice Log Messages The informative log messages include:

`md: d`*num*`: Hotspared device` *dev* `with` *dev*

The first device name listed has been hot spare replaced with the second device name listed.

`md: d`*num*`: Hotspared device dev(`*num*`,`*num*`) with dev(`*num*`,`*num*`)`

The first device number listed has been hot spare replaced with the second device number listed.

`md: Could not load misc /`*dev*

The named `misc` module is not loadable. It is possibly missing, or something else has been copied over it.

`md: d`*num*`: no mem for property` *dev*

Memory could not be allocated in the `prop_op` entry point.

`md: db: Parsing error on '`*dev*`'`

Set command in `/kernel/drv/md.conf` for the `mddb.bootlist` <*number*> is not in the correct format. `metadb -p` can be run to put the correct set commands into the `/kernel/drv/md.conf` file.

`md: d`*num*`:` *dev*`(`*num*`,`*num*`) needs maintenance`
`md: d`*num*`:` *dev* `needs maintenance`

An I/O or open error has occurred on a device within a mirror causing a component in the mirror to change to the Maintenance state.

`md: d`*num*`:` *dev*`(`*num*`,`*num*`) last erred md: d`*num*`:` *dev* `last erred`

An I/O or open error has occurred on a device within a mirror and the data is not replicated elsewhere in the mirror. This is causing the component in the mirror to change to the Last Erred state.

Warning Log Messages The warning log messages include:

`md: State database is stale`

This error message comes when there are not enough usable replicas for the state database to be able to update records in the database. All accesses to the metadevice driver will fail. To fix this problem, more replicas need to be added or inaccessible replicas need to be deleted.

`md: d`*num*`: read error on` *dev*`md: d`*num*`: write error on` *dev*

A read or write error has occurred on the specified submirror, at the specified device name. This happens if any read or write errors occur on a submirror.

`md: d`*num*`: read error on dev(`*num*`,`*num*`)md: d`*num*`: write error on dev(` *num*`,`*num*`)`

A read or write error has occurred on the specified submirror, at the specified device number. This happens if any read or write errors occur on a submirror.

```
md: State database commit failed
md: State database delete failed
```

These messages occur when there have been device errors on components where the state database replicas reside. These errors only occur when more than half of the replicas have had device errors returned to them. For instance, if you have three components with state database replicas and two of the components report errors, then these errors may occur. The state database commit or delete is retried periodically. If a replica is added, then the commit or delete will finish and the system will be operational. Otherwise the system will timeout and panic.

`md: d`*num*`: Cannot load` *dev* `driver`

Underlying named driver module is not loadable (for example, `sd`, `id`, `xy`, or a third-party driver). This could indicate that the driver module has been removed.

`md: Open error of hotspare` *dev*`md: Open error of hotspare dev(`*num*`,`*num*`)`

Named hotspare is not openable, or underlying driver is not loadable.

Panic Log Messages

The panic log messages include:

`md: d`*num*`: Unknown close typemd: d`*num*`: Unknown open type`

Metadevice is being opened/closed with an unknown open type (OTYP).

`md: State database problem`

Failed metadevice state database commit or delete has been retried the default 100 times.

Files

`/dev/md/dsk/d`*n*	block device (where *n* is the device number)
`/dev/md/rdsk/d`*n*	raw device (where *n* is the device number)
`/dev/md/`*setname*`/dsk/d`*n*	block device (where *setname* is the name of the diskset and *n* is the device number)
`/dev/md/`*setname*`/rdsk/d`*n*	raw device (where *setname* is the name of the diskset and *n* is the device number)
`/dev/md/admin`	administrative device

`/kernel/drv/md`	driver module
`/kernel/drv/md.conf`	driver configuration file
`/kernel/misc/md_stripe`	stripe driver misc module
`/kernel/misc/md_mirror`	mirror driver misc module
`/kernel/misc/md_hotspares`	hotspares driver misc module
`/kernel/misc/md_trans`	metatrans driver for UFS logging
`/kernel/misc/md_raid`	RAID5 driver misc module

See Also `mdmonitord`(1M), `metaclear`(1M), `metadb`(1M), `metadetach`(1M), `metahs`(1M), `metainit`(1M), `metaoffline`(1M), `metaonline`(1M), `metaparam`(1M), `metarecover`(1M), `metarename`(1M), `metareplace`(1M), `metaroot`(1M), `metassist`(1M), `metaset`(1M), `metastat`(1M), `metasync`(1M), `metattach`(1M), `md.cf`(4), `md.tab`(4), `attributes`(5),

Solaris Volume Manager Administration Guide

Notes Trans metadevices have been replaced by UFS logging. Existing trans devices are *not* logging--they pass data directly through to the underlying device. See `mount_ufs`(1M) for more information about UFS logging.

Name mediator – support for HA configurations consisting of two strings of drives

Description Beginning with a prior version, Solaris Volume Manager provided support for high-availability (HA) configurations consisting of two hosts that share at least three strings of drives and that run software enabling exclusive access to the data on those drives from one host. (Note: Volume Manager, by itself, does not actually provide a high-availability environment. The diskset feature is an enabler for HA configurations.)

Volume Manager provides support for a low-end HA solution consisting of two hosts that share only two strings of drives. The hosts in this type of configuration, referred to as *mediators*, run a special daemon, `rpc.metamedd(1M)`. The mediator hosts take on additional responsibilities to ensure that data is available in the case of host or drive failures.

In a mediator configuration, two hosts are physically connected to two strings of drives. This configuration can survive the failure of a single host or a single string of drives, without administrative intervention. If both a host and a string of drives fail (multiple failures), the integrity of the data cannot be guaranteed. At this point, administrative intervention is required to make the data accessible.

The following definitions pertain to a mediator configuration:

`diskset`	A set of drives containing metadevices and hot spares that can be shared exclusively (but not concurrently) by two hosts.
`Volume Manager state database`	A replicated database that stores metadevice configuration and state information.
`mediator host`	A host that runs the `rpc.metamedd(1M)` daemon and that has been added to a diskset. The mediator host participates in checking the state database and the mediator quorum.
`mediator quorum`	The condition achieved when the number of accessible mediator hosts is equal to half+1 the total number of configured mediator hosts. Because it is expected that there will be two mediator hosts, this number will normally be 2 ([(2/2) + 1] = 2.)
`replica`	A single copy of the Volume Manager metadevice state database.
`replica quorum`	The condition achieved when the number of accessible replicas is equal to half+1 the total number of configured replicas. For example, if a system is configured with ten replicas, the quorum is met when six are accessible ([(10/2) + 1 = 6]).

A mediator host running the `rpc.metamedd(1M)` daemon keeps track of replica updates. As long as the following conditions are met, access to data occurs without any administrative intervention:

- The replica quorum is not met.
- Half of the replicas are still accessible.
- The mediator quorum is met.

The following conditions describe the operation of mediator hosts:

1. If the is met, access to the diskset is granted. At this point no mediator host is involved.
2. If the replica quorum is not met, half of the replicas are accessible, the mediator quorum is met, and the replica and mediator data match, access to the diskset is granted. The mediator host contributes the deciding vote.
3. If the replica quorum is not met, half of the replicas are accessible, the mediator quorum is not met, half of the mediator hosts is accessible, and the replica and mediator data match, the system prompts you to grant or deny access to the diskset.
4. If the replica quorum is not met, half of the replicas are accessible, the mediator quorum is met, and the replica and mediator data do not match, access to the diskset is read-only. You can delete replicas, release the diskset, and retake the diskset to gain read-write access to the data in the diskset.
5. In all other cases, the diskset access is read-only. You can delete replicas, release the diskset, and retake the diskset to gain read-write access to the data in the diskset.

The `metaset(1M)` command administers disksets and mediator hosts. The following options to the `metaset` command pertain only to administering mediator hosts.

`-a -m` *mediator_host_list*	Adds mediator hosts to the named set. A *mediator_host_list* is the nodename of the mediator host to be added and up to 2 other aliases for the mediator host. The nodename and aliases for each mediator host are separated by commas. Up to 3 mediator hosts can be specified for the named diskset.
`-d -m` *mediator_host_list*	Deletes mediator hosts from the named diskset. Mediator hosts are deleted from the diskset by specifying the nodename of mediator host to delete.
`-q`	Displays an enumerated list of tags pertaining to “tagged data" that may be encountered during a take of the ownership of a diskset.
`-t [-f] -y`	Takes ownership of a diskset safely, unless `-f` is used, in which case the take is unconditional. If `metaset` finds that another host owns the set, this host will not be allowed to take ownership of the set. If the set is not owned by any other host, all the disks within the set will be owned by the host on which `metaset` was

executed. The metadevice state database is read in and the shared metadevices contained in the set become accessible. The `-t` option will take a diskset that has stale databases. When the databases are stale, `metaset` will exit with code 66, and a message will be printed. At that point, the only operations permitted are the addition and deletion of replicas. Once the addition or deletion of the replicas has been completed, the diskset should be released and retaken to gain full access to the data. If mediator hosts have been configured, some additional exit codes are possible. If half of the replicas and half of the mediator hosts are operating properly, the take will exit with code 3. At this point, you can add or delete replicas, or use the `-y` option on a subsequent take. If the take operation encounters "tagged data," the take operation will exit with code 2. You can then run the `metaset` command with the `-q` option to see an enumerated list of tags.

`-t` [`-f`] `-u` *tagnumber* — Once a tag has been selected, a subsequent take with `-u` *tagnumber* can be executed to select the data associated with the given *tagnumber*.

See Also `metaset(1M)`, `md(7D)`, `rpc.metamedd(1M)`, `rpc.metad(1M)`

Sun Cluster documentation, *Solaris Volume Manager Administration Guide*

Notes Diskset administration, including the addition and deletion of hosts and drives, requires all hosts in the set to be accessible from the network.

Name mega_sas – SCSI HBA driver for LSI MegaRAID SAS controller

Description The `mega_sas` MegaRAID controller host bus adapter driver is a SCSA-compliant nexus driver that supports the Dell PERC 5/E, 5/i, 6/E and 6/i RAID controllers, the IBM ServeRAID-MR10k SAS/SATA controller and the LSI MegaRAID SAS/SATA 8308ELP, 8344ELP, 84016E, 8408ELP, 8480ELP, 8704ELP, 8704EM2, 8708ELP, 8708EM2, 8880EM2 and 8888ELP series of controllers.

Supported RAID features include RAID levels 0, 1, 5, and 6, RAID spans 10, 50 and 60, online capacity expansion (OCE), online RAID level migration (RLM), auto resume after loss of system power during arrays, array rebuild or reconstruction (RLM) and configurable stripe size up to 1MB. Additional supported RAID features include check consistency for background data integrity, patrol read for media scanning and repairing, 64 logical drive support, up to 64TB LUN support, automatic rebuild and global and dedicated hot spare support.

Configuration The `mega_sas.conf` file contains no user configurable parameters. Please configure your hardware through the related BIOS utility or the MegaCli configuration utility. If you want to install to a drive attached to a mega_sas HBA, you should create the virtual drive first from the BIOS before running the Solaris install. You can obtain the MegaCli utility from the LSI website.

The `mega_sas` device can support up to 64 virtual disks. Note that BIOS numbers the virtual disks as 1 through 64, however in the Solaris operating environment virtual disks are numbered from 0 to 63. Also note that SAS and SATA drives cannot be configured into the same virtual disk.

Known Problems and Limitations The `mega_sas` driver does not support the LSI MegaRAID SAS 8204ELP, 8204XLP, 8208ELP, and 8208XLP controllers.

Files

`/kernel/drv/mega_sas`	32-bit ELF kernel module. (x86)
`/kernel/drv/amd64/mega_sas`	64-bit kernel module. (x86)
`/kernel/drv/mega_sas.conf`	Driver configuration file (contains no user-configurable options).

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86-based systems
Availability	SUNWmegasas
Interface stability	Uncommitted

See Also prtconf(1M), attributes(5), sata(7D), scsi_hba_attach_setup(9F), scsi_sync_pkt(9F), scsi_transport(9F), scsi_inquiry(9S), scsi_device(9S), scsi_pkt(9S)

Small Computer System Interface-2 (SCSI-2)

Name mem, kmem, allkmem – physical or virtual memory access

Synopsis

```
/dev/mem

/dev/kmem

/dev/allkmem
```

Description The file `/dev/mem` is a special file that provides access to the physical memory of the computer.

The file `/dev/kmem` is a special file that provides access to the virtual address space of the operating system kernel, excluding memory that is associated with an I/O device.

The file `/dev/allkmem` is a special file that provides access to the virtual address space of the operating system kernel, including memory that is associated with an I/O device. You can use any of these devices to examine and modify the system.

Byte addresses in `/dev/mem` are interpreted as physical memory addresses. Byte addresses in `/dev/kmem` and `/dev/allkmem` are interpreted as kernel virtual memory addresses. A reference to a non-existent location returns an error. See ERRORS for more information.

The file `/dev/mem` accesses physical memory; the size of the file is equal to the amount of physical memory in the computer. This size may be larger than 4GB on a system running the 32-bit operating environment. In this case, you can access memory beyond 4GB using a series of `read(2)` and `write(2)` calls, a `pread64()` or `pwrite64()` call, or a combination of `llseek(2)` and `read(2)` or `write(2)`.

Errors

`EFAULT` Occurs when trying to `write(2)` a read-only location (`allkmem`), `read(2)` a write-only location (`allkmem`), or `read(2)` or `write(2)` a non-existent or unimplemented location (`mem`, `kmem`, `allkmem`).

`EIO` Occurs when trying to `read(2)` or `write(2)` a memory location that is associated with an I/O device using the `/dev/kmem` special file.

`ENXIO` Results from attempting to `mmap(2)` a non-existent physical (`mem`) or virtual (`kmem`, `allkmem`) memory address.

Files

`/dev/mem` Provides access to the computer's physical memory.

`/dev/kmem` Provides access to the virtual address space of the operating system kernel, excluding memory that is associated with an I/O device.

`/dev/allkmem` Provides access to the virtual address space of the operating system kernel, including memory that is associated with an I/O device.

See Also `llseek(2)`, `mmap(2)`, `read(2)`, `write(2)`

Warnings Using these devices to modify (that is, write to) the address space of a live running operating system or to modify the state of a hardware device is extremely dangerous and may result in a system panic if kernel data structures are damaged or if device state is changed.

Name mhd – multihost disk control operations

Synopsis

```
#include <sys/mhd.h>
```

Description The `mhd` `ioctl(2)` control access rights of a multihost disk, using disk reservations on the disk device.

The stability level of this interface (see `attributes(5)`) is evolving. As a result, the interface is subject to change and you should limit your use of it.

The mhd ioctls fall into two major categories: (1) ioctls for non-shared multihost disks and (2) ioctls for shared multihost disks.

One ioctl, `MHIOCENFAILFAST`, is applicable to both non-shared and shared multihost disks. It is described after the first two categories.

All the ioctls require root privilege.

For all of the ioctls, the caller should obtain the file descriptor for the device by calling `open(2)` with the `O_NDELAY` flag; without the `O_NDELAY` flag, the open may fail due to another host already having a conflicting reservation on the device. Some of the ioctls below permit the caller to forcibly clear a conflicting reservation held by another host, however, in order to call the ioctl, the caller must first obtain the open file descriptor.

Non-shared multihost disks

Non-shared multihost disks ioctls consist of `MHIOCTKOWN`, `MHIOCRELEASE`, `HIOCSTATUS`, and `MHIOCQRESERVE`. These ioctl requests control the access rights of non-shared multihost disks. A non-shared multihost disk is one that supports serialized, mutually exclusive I/O mastery by the connected hosts. This is in contrast to the shared-disk model, in which concurrent access is allowed from more than one host (see below).

A non-shared multihost disk can be in one of two states:

- Exclusive access state, where only one connected host has I/O access
- Non-exclusive access state, where all connected hosts have I/O access. An external hardware reset can cause the disk to enter the non-exclusive access state.

Each multihost disk driver views the machine on which it's running as the "local host"; each views all other machines as "remote hosts". For each I/O or ioctl request, the requesting host is the local host.

Note that the non-shared ioctls are designed to work with SCSI-2 disks. The SCSI-2 RESERVE/RELEASE command set is the underlying hardware facility in the device that supports the non-shared ioctls.

The function prototypes for the non-shared ioctls are:

```
ioctl(fd, MHIOCTKOWN);
ioctl(fd, MHIOCRELEASE);
```

```
ioctl(fd, MHIOCSTATUS);
ioctl(fd, MHIOCQRESERVE);
```

`MHIOCTKOWN` Forcefully acquires exclusive access rights to the multihost disk for the local host. Revokes all access rights to the multihost disk from remote hosts. Causes the disk to enter the exclusive access state.

Implementation Note: Reservations (exclusive access rights) broken via random resets should be reinstated by the driver upon their detection, for example, in the automatic probe function described below.

`MHIOCRELEASE` Relinquishes exclusive access rights to the multihost disk for the local host. On success, causes the disk to enter the non- exclusive access state.

`MHIOCSTATUS` Probes a multihost disk to determine whether the local host has access rights to the disk. Returns `0` if the local host has access to the disk, `1` if it doesn't, and `-1` with errno set to `EIO` if the probe failed for some other reason.

`MHIOCQRESERVE` Issues, simply and only, a SCSI-2 Reserve command. If the attempt to reserve fails due to the SCSI error Reservation Conflict (which implies that some other host has the device reserved), then the ioctl will return –1 with errno set to `EACCES`. The `MHIOCQRESERVE` ioctl does NOT issue a bus device reset or bus reset prior to attempting the SCSI-2 reserve command. It also does not take care of re-instating reservations that disappear due to bus resets or bus device resets; if that behavior is desired, then the caller can call `MHIOCTKOWN` after the `MHIOCQRESERVE` has returned success. If the device does not support the SCSI-2 Reserve command, then the ioctl returns –1 with `errno` set to `ENOTSUP`. The `MHIOCQRESERVE` ioctl is intended to be used by high-availability or clustering software for a "quorum" disk, hence, the "Q" in the name of the ioctl.

Shared Multihost Disks

Shared multihost disks ioctls control access to shared multihost disks. The ioctls are merely a veneer on the SCSI-3 Persistent Reservation facility. Therefore, the underlying semantic model is not described in detail here, see instead the SCSI-3 standard. The SCSI-3 Persistent Reservations support the concept of a group of hosts all sharing access to a disk.

The function prototypes and descriptions for the shared multihost ioctls are as follows:

`ioctl(fd, MHIOCGRP_INKEYS, (mhioc_inkeys_t) *k);`

Issues the SCSI-3 command Persistent Reserve In Read Keys to the device. On input, the field `k->li` should be initialized by the caller with `k->li.listsize` reflecting how big of an array the caller has allocated for the `k->li.list` field and with `k->li.listlen == 0`. On return, the field `k->li.listlen` is updated to indicate the number of reservation keys the device currently has: if this value is larger than `k->li.listsize` then that indicates that the caller should have passed a bigger `k->li.list` array with a bigger `k->li.listsize`. The number of array elements actually written by the callee into `k->li.list` is the minimum of

k->li.listlen and k->li.listsize. The field k->generation is updated with the generation information returned by the SCSI-3 Read Keys query. If the device does not support SCSI-3 Persistent Reservations, then this ioctl returns –1 with errno set to ENOTSUP.

ioctl(fd, MHIOCGRP_INRESV, (mhioc_inresvs_t) *r);
Issues the SCSI-3 command Persistent Reserve In Read Reservations to the device. Remarks similar to MHIOCGRP_INKEYS apply to the array manipulation. If the device does not support SCSI-3 Persistent Reservations, then this ioctl returns –1 with errno set to ENOTSUP.

ioctl(fd, MHIOCGRP_REGISTER, (mhioc_register_t) *r);
Issues the SCSI-3 command Persistent Reserve Out Register. The fields of structure *r* are all inputs; none of the fields are modified by the ioctl. The field r->aptpl should be set to true to specify that registrations and reservations should persist across device power failures, or to false to specify that registrations and reservations should be cleared upon device power failure; true is the recommended setting. The field r->oldkey is the key that the caller believes the device may already have for this host initiator; if the caller believes that that this host initiator is not already registered with this device, it should pass the special key of all zeros. To achieve the effect of unregistering with the device, the caller should pass its current key for the r->oldkey field and an r->newkey field containing the special key of all zeros. If the device returns the SCSI error code Reservation Conflict, this ioctl returns –1 with errno set to EACCES.

ioctl(fd, MHIOCGRP_RESERVE, (mhioc_resv_desc_t) *r);
Issues the SCSI-3 command Persistent Reserve Out Reserve. The fields of structure *r* are all inputs; none of the fields are modified by the ioctl. If the device returns the SCSI error code Reservation Conflict, this ioctl returns –1 with errno set to EACCES.

ioctl(fd, MHIOCGRP_PREEMPTANDABORT, (mhioc_preemptandabort_t) *r);
Issues the SCSI-3 command Persistent Reserve Out Preempt-And-Abort. The fields of structure *r* are all inputs; none of the fields are modified by the ioctl. The key of the victim host is specified by the field r->victim_key. The field r->resvdesc supplies the preempter's key and the reservation that it is requesting as part of the SCSI-3 Preempt-And-Abort command. If the device returns the SCSI error code Reservation Conflict, this ioctl returns –1 with errno set to EACCES.

ioctl(fd, MHIOCGRP_PREEMPT, (mhioc_preemptandabort_t) *r);
Similar to MHIOCGRP_PREEMPTANDABORT, but instead issues the SCSI-3 command Persistent Reserve Out Preempt. (Note: This command is not implemented).

ioctl(fd, MHIOCGRP_CLEAR, (mhioc_resv_key_t) *r);
Issues the SCSI-3 command Persistent Reserve Out Clear. The input parameter *r* is the reservation key of the caller, which should have been already registered with the device, by an earlier call to MHIOCGRP_REGISTER. (Note: This command is not implemented).

For each device, the non-shared ioctls should not be mixed with the Persistent Reserve Out shared ioctls, and vice-versa, otherwise, the underlying device is likely to return errors,

because SCSI does not permit SCSI-2 reservations to be mixed with SCSI-3 reservations on a single device. It is, however, legitimate to call the Persistent Reserve In ioctls, because these are query only. Issuing the `MHIOCGRP_INKEYS` ioctl is the recommended way for a caller to determine if the device supports SCSI-3 Persistent Reservations (the ioctl will return –1 with `errno` set to `ENOTSUP` if the device does not).

MHIOCENFAILFAST ioctl

The `MHIOCENFAILFAST` ioctl is applicable for both non-shared and shared disks, and may be used with either the non-shared or shared ioctls.

`ioctl(fd, MHIOENFAILFAST, (unsigned int *)` *millisecs*`);`

Enables or disables the failfast option in the multihost disk driver and enables or disables automatic probing of a multihost disk, described below. The argument is an unsigned integer specifying the number of milliseconds to wait between executions of the automatic probe function. An argument of zero disables the failfast option and disables automatic probing. If the `MHIOCENFAILFAST` ioctl is never called, the effect is defined to be that both the failfast option and automatic probing are disabled.

Automatic Probing

The `MHIOCENFAILFAST` ioctl sets up a timeout in the driver to periodically schedule automatic probes of the disk. The automatic probe function works in this manner: The driver is scheduled to probe the multihost disk every n milliseconds, rounded up to the next integral multiple of the system clock's resolution. If

1. the local host no longer has access rights to the multihost disk, and
2. access rights were expected to be held by the local host,

the driver immediately panics the machine to comply with the failfast model.

If the driver makes this discovery outside the timeout function, especially during a read or write operation, it is imperative that it panic the system then as well.

Return Values

Each request returns –1 on failure and sets `errno` to indicate the error.

`EPERM`	Caller is not root.
`EACCES`	Access rights were denied.
`EIO`	The multihost disk or controller was unable to successfully complete the requested operation.
`EOPNOTSUP`	The multihost disk does not support the operation. For example, it does not support the SCSI-2 Reserve/Release command set, or the SCSI-3 Persistent Reservation command set.

Attributes

See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWhea

Stability	Evolving

See Also ioctl(2), open(2), attributes(5), open(2)

Name mixer – generic mixer device interface

Synopsis

```
#include <sys/soundcard.h>
```

Description .

Mixer Pseudo-Device The `/dev/mixer` pseudo-device is provided for two purposes:

- The first purpose is for applications that wish to learn about the list of audio devices on the system, so that they can select (or provide for users to select) an appropriate audio device. The `/dev/mixer` pseudo-device provides interfaces to enumerate all of the audio devices on the system.
- The second purpose is for mixer panel type applications which need to control master settings for the audio hardware in the system, such as gain levels, balance, port functionality, and other device features.

Ordinary audio applications should *not* attempt to adjust their playback or record volumes or other device settings using this device. Instead, they should use the `SNDCTL_DSP_SETPLAYVOL` and `SNDCTL_DSP_SETRECVOL` ioctls that are documented in `dsp(7I)`.

Sndstat Device The `/dev/sndstat` device supports `read(2)`, and can be read to retrieve human-readable information about the audio devices on the system. Software should not attempt to interpret the contents of this device.

ioctls

Information IOCTLs The following ioctls are intended to aid applications in identifying the audio devices available on the system. These ioctls can be issued against either the pseudo-device `/dev/mixer`, or a against a file descriptor open to any other audio device in the system.

Applications should issue `SNDCTL_SYSINFO` first to learn what audio devices and mixers are available on the system, and then use `SNDCTL_AUDIOINFO` or `SNDCTL_MIXERINFO` to obtain more information about the audio devices or mixers, respectively.

`OSS_GETVERSION` The argument is a pointer to an integer, which retrieves the version of the `OSS API` used. The value is encoded with the major version (currently 4) encoded in the most significant 16 bits, and a minor version encoded in the lower 16 bits.

`SNDCTL_SYSINFO` The argument is a pointer to an `oss_sysinfo` structure, which has the following definition:

```
typedef struct oss_sysinfo {
   char product[32];   /* E.g. SunOS Audio */
   char version[32];   /* E.g. 4.0a */
   int versionnum;     /* See OSS_GETVERSION */
   char options[128];  /* NOT SUPPORTED */

   int numaudios;      /* # of audio/dsp devices */
```

```
  int openedaudio[8]; /* Reserved, always 0 */

 int numsynths;        /* NOT SUPPORTED, always 0 */
 int nummidis;         /* NOT SUPPORTED, always 0 */
 int numtimers;        /* NOT SUPPORTED, always 0 */
 int nummixers;        /* # of mixer devices */

 int openedmidi[8];    /* Mask of midi devices are
                          busy */
 int numcards;         /* Number of sound cards in
                          the system */
 int numaudioengines;  /* Number of audio engines in
                          the system */
 char license[16];     /* E.g. "GPL" or "CDDL" */
 char revision_info[256];  /* Reserved */
 int filler[172];          /* Reserved */
} oss_sysinfo;
```

The important fields here are `numaudios`, which is used to determine the number of audio devices that can be queried with `SNDCTL_AUDIOINFO`, `nummixers` which provides a count of mixers on the system, and `numcards` which counts to total number of aggregate devices. A `card` can consist of one or more audio devices and one or more mixers, although more typically there is exactly one audio device and one mixer for each card.

`SNDCTL_AUDIOINFO` The argument is a pointer to an `oss_audioinfo` structure, which has the following structure:

```
typedef struct oss_audioinfo {
   int dev;  /* Device to query */
   char name[64];  /* Human readable name */
   int busy;  /* reserved */
   int pid;  /* reserved */
   int caps;  /* PCM_CAP_INPUT, PCM_CAP_OUTPUT */
   int iformats;  /* Supported input formats */
   int oformats;  /* Supported output formats */
   int magic;  /* reserved */
   char cmd[64];  /* reserved */
   int card_number;
   int port_number;  /* reserved */
   int mixer_dev;
   int legacy_device; /* Obsolete field.
                         Replaced by devnode */
   int enabled;  /* reserved */
   int flags;  /* reserved */
   int min_rate;  /* Minimum sample rate */
   int max_rate;  /* Maximum sample rate */
```

```
    int min_channels;  /* Minimum number
                          of channels */
    int max_channels;  /* Maximum number
                          of channels */
    int binding;  /* reserved */
    int rate_source;  /* reserved */
    char handle[32];  /* reserved */
    unsigned int nrates;  /* reserved */
    unsigned int rates[20];  /* reserved */
    char song_name[64];  /* reserved */
    char label[16];  /* reserved */
    int latency;  /* reserved */
    char devnode[32];  /* Device special file
                          name (absolute path) */
    int next_play_engine;  /* reserved */
    int next_rec_engine;  /* reserved */
    int filler[184];  /* reserved */
} oss_audioinfo;
```

In the above structure, all of the fields are reserved except the following: `dev`, `name`, `card_number`, `mixer_dev`, `caps`, `min_rate`, `max_rate`, `min_channels`, `max_channels`, and `devnode`. The reserved fields are provided for compatibility with other OSS implementations, and available for legacy applications. New applications should not attempt to use these fields.

The `dev` field should be initialized by the application to the number of the device to query. This is a number between zero (inclusive) and value of `numaudios` (exclusive) returned by `SNDCTL_SYSINFO`. Alternatively, when issuing the ioctl against a real mixer or `dsp` device, the special value `-1` can be used to indicate that the query is being made against the device opened. If `-1` is used, the field is overwritten with the device number for the current device on successful return.

No other fields are significant upon entry, but a successful return contains details of the device.

The `name` field is a human readable name representing the device. Applications should not try to interpret it.

The `card_number` field indicates the number assigned to the aggregate device. This can be used with the `SNDCTL_CARDINFO` ioctl.

The `mixer_dev` is the mixer device number for the mixing device associated with the audio device. This can be used with the `SNDCTL_MIXERINFO` ioctl.

The caps field contains any of the bits PCM_CAP_INPUT, PCM_CAP_OUTPUT, and PCM_CAP_DUPLEX. Indicating whether the device support input, output, and whether input and output can be used simultaneously. All other bits are reserved.

The min_rate and max_rate fields indicate the minimum and maximum sample rates supported by the device. Most applications should try to use the maximum supported rate for the best audio quality and lowest system resource consumption.

The min_channels and max_channels provide an indication of the number of channels (1 for mono, 2 for stereo, 6 for 5.1, etc.) supported by the device.

The devnode field contains the actual full path to the device node for this device, such as /dev/sound/audio810:0dsp. Applications should open this file to access the device.

SNDCTL_CARDINFO The argument is a pointer to a struct oss_card_info, which has the following definition:

```
typedef struct oss_card_info {
int card;
 char shortname[16];
 char longname[128];
 int flags;/* reserved */
 char hw_info[400];
 int intr_count;/* reserved */
 int ack_count;/* reserved */
 int filler[154];
} oss_card_info;
```

This ioctl is used to query for information about the aggregate audio device.

The card field should be initialized by the application to the number of the card to query. This is a number between zero inclusive and value of numcards (exclusive) returned by SNDCTL_SYSINFO.) Alternatively, when issuing the ioctl against a real mixer or dsp device, the special value -1 can be used to indicate that the query is being made against the device opened. If -1 is used, the field is overwritten with the number for the current hardware device on successful return.

The shortname, longname, and hw_info contain ASCIIZ strings describing the device in more detail. The hw_info member can contain multiple lines of detail, each line ending in a NEWLINE.

The flag, intr_count, and ack_count fields are not used by this implementation.

`SNDCTL_MIXERINFO` The argument is a pointer to a `struct oss_mixer_info`, which has the following definition:

```
typedef struct oss_mixerinfo {
  int dev;
  char id[16];/* Reserved */
  char name[32];
  int modify_counter;
  int card_number;
  int port_number;/* Reserved */
  char handle[32];/* Reserved */
  int magic;/* Reserved */
  int enabled;/* Reserved */
  int caps;/* Reserved */
  int flags;/* Reserved */
  int nrext;
  int priority;
  char devnode[32];/* Device special file name
                   (absolute path) */
  int legacy_device;/* Reserved */
  int filler[245];/* Reserved */
} oss_mixerinfo;
```

In the above structure, all of the fields are reserved except the following: `dev, name, modify_counter, card_number, nrext, priority,` and `devnode`. The reserved fields are provided for compatibility with other OSS implementations, and available for legacy applications. New applications should not attempt to use these fields.

The `dev` field should be initialized by the application to the number of the device to query. This is a number between zero inclusive and value of `nummixers` (exclusive) returned by `SNDCTL_SYSINFO`, or by `SNDCTL_MIX_NRMIX`. Alternatively, when issuing the ioctl against a real mixer or `dsp` device, the special value `-1` can be used to indicate that the query is being made against the device opened. If `-1` is used, the field is overwritten with the mixer number for the current open file on successful return.

No other fields are significant upon entry, but on successful return contains details of the device.

The `name` field is a human readable name representing the device. Applications should not try to interpret it.

The `modify_counter` is changed by the mixer framework each time the settings for the various controls or extensions of the device are changed. Applications can poll this value to learn if any other changes need to be searched for.

The `card_number` field is the number of the aggregate audio device this mixer is located on. It can be used with the `SNDCTL_CARDINFO` ioctl.

The `nrext` field is the number of mixer extensions available on this mixer. See the `SNDCTL_MIX_NREXT` description.

The priority is used by the framework to assign a preference that applications can use in choosing a device. Higher values are preferable. Mixers with priorities less than -1 should never be selected by default.

The `devnode` field contains the actual full path to the device node for the physical mixer, such as `/dev/sound/audio810:0mixer`. Applications should open this file to access the mixer settings.

Mixer Extension IOCTLs The pseudo `/dev/mixer` device supports ioctls that can change the various settings for the audio hardware in the system.

Those ioctls should only be used by dedicated mixer applications or desktop volume controls, and not by typical ordinary audio applications such as media players. Ordinary applications that wish to adjust their own volume settings should use the `SNDCTL_DSP_SETPLAYVOL` or `SNDCTL_DSP_SETRECVOL` ioctls for that purpose. See `dsp`(7I) for more information. Ordinary applications should never attempt to change master port selection or hardware settings such as monitor gain settings.

The ioctls in this section can only be used to access the mixer device that is associated with the current file descriptor.

Applications should not assume that a single `/dev/mixer` node is able to access any physical settings. Instead, they should use the ioctl `SNDCTL_MIXERINFO` to determine the device path for the real mixer device, and issue ioctls on a file descriptor opened against the corresponding `devnode` field.

When a `dev` member is specified in each of the following ioctls, the application should specify `-1`, although for compatibility the mixer allows the application to specify the mixer device number.

`SNDCTL_MIX_NRMIX` The argument is a pointer to an integer, which receives the number of mixer devices in the system. Each can be queried by

using its number with the SNDCTL_MIXERINFO ioctl. The same information is available using the SNDCTL_SYSINFO ioctl.

SNDCTL_MIX_NREXT

The argument is a pointer to an integer. On entry, the integer should contain the special value -1. On return the argument receives the number of mixer extensions (or mixer controls) supported by the mixer device. More details about each extension can be obtained by SNDCTL_MIX_EXTINFO ioctl.

SNDCTL_MIX_EXTINFO

The argument is a pointer to an oss_mixext structure which is defined as follows:

```
typedef struct oss_mixext {
   int dev;  /* Mixer device number */
   int ctrl;  /* Extension number */
   int type;  /* Entry type */
   int maxvalue;
   int minvalue;
   int flags;
   char id[16];  /* Mnemonic ID (internal use) */
   int parent;   /* Entry# of parent
                    (-1 if root) */
   int dummy;   /* NOT SUPPORTED */
   int timestamp;
   char data[64];  /* Reserved */
   unsigned char enum_present[32];  /* Mask
                                       of allowed
                                       enum
                                       values */
   int control_no;  /* Reserved */
   unsigned int desc;  /* NOT SUPPORTED */
   char extname[32];
   int update_counter;
   int filler[7];  /* Reserved */
} oss_mixext;
```

On entry, the dev field should be initialized to the value -1, and the ctrl field should be initialized with the number of the extension being accessed. Between 0, inclusive, and the value returned by SNDCTL_MIX_NREXT, exclusive.

Mixer extensions are organized as a logical tree, starting with a root node. The root node always has a ctrl value of zero. The structure of the tree can be determined by looking at the parent field, which contains the extension number of the parent extension, or -1 if the extension is the root extension.

The type indicates the type of extension used. This implementation supports the following values:

```
MIXT_DEVROOT      Root node for extension tree
MIXT_GROUP        Logical grouping of controls
MXIT_ONOFF        Boolean value, 0 = off, 1 = on.
MIXT_ENUM         Enumerated value, 0 to maxvalue.
MIXT_MONOSLIDER   Monophonic slider, 0 to 255.
MIXT_STEREOSLIDER Stereophonic slider, 0 to 255
                  (encoded as
                  lower two bytes in value.)
MIXT_MARKER       Place holder, can ignore.
```

The flags field is a bit array. This implementation makes use of the following possible bits:

```
MIXF_READABLE    Extension's value is readable.
MIXF_WRITEABLE   Extension's value is modifiable.
MIXF_POLL        Extension can self-update.
MIXF_PCMVOL      Extension is for master
                 PCM playback volume.
MIXF_MAINVOL     Extension is for a typical
                 analog volume
MIXF_RECVOL      Extension is for master
                 record gain.
MIXF_MONVOL      Extension is for a monitor
                 source's gain.
```

The `id` field contains an `ASCIIZ` identifier for the extension.

The timestamp field is set when the extension tree is first initialized. Applications must use the same timestamp value when attempting to change the values. A change in the timestamp indicates a change a in the structure of the extension tree.

The `enum_present` field is a bit mask of possible enumeration values. If a bit is present in the `enum_present` mask, then the corresponding enumeration value is legal. The mask is in little endian order.

The `desc` field provides information about scoping, which can be useful as layout hints to applications. The following hints are available:

```
MIXEXT_SCOPE_MASK    Mask of possible scope
                     values.
MIXEXT_SCOPE_INPUT   Extension is an input
                     control.
MIXEXT_SCOPE_OUTPUT  Extension is an
```

```
                        output control.
MIXEXT_SCOPE_MONITOR Extension relates to
                        input monitoring.
MIXEXT_SCOPE_OTHER   No scoping hint provided.
```

The `extname` is the full name of the extension.

The `update_counter` is incremented each time the control's value is changed.

`SNDCTL_MIX_ENUMINFO` The argument is a pointer to an `oss_mixer_enuminfo` structure, which is defined as follows:

```
typedef struct oss_mixer_enuminfo {
   int dev;
   int ctrl;
   int nvalues;
   int version;
   short strindex[255];
   char strings[3000];
} oss_mixer_enuminfo;
```

On entry, the `dev` field should be initialized to the value `-1`, and the `ctrl` field should be initialized with the number of the extension being accessed. Between 0, inclusive, and the value returned by `SNDCTL_MIX_NREXT`, exclusive.

On return the `nvalues` field contains the number of values, and `strindex` contains an array of indices into the strings member, each index pointing to an `ASCIIZ` describing the enumeration value.

`SNDCTL_MIX_READ`
`SNDCTL_MIX_WRITE` The argument is a pointer to an `oss_mixer_value` structure, defined as follows:

```
typedef struct oss_mixer_value {
   int dev;
   int ctrl;
   int value;
   int flags;  /* Reserved for future use.
                  Initialize to 0 */
   int timestamp;  /* Must be set to
                      oss_mixext.timestamp */
   int filler[8];  /* Reserved for future use.
                      Initialize to 0 */
} oss_mixer_value;
```

On entry, the dev field should be initialized to the value -1, and the ctrl field should be initialized with the number of the extension being accessed. Between 0, inclusive, and the value returned by SNDCTL_MIX_NREXT, exclusive. Additionally, the timestamp member must be initialized to the same value as was supplied in the oss_mixext structure used with SNDCTL_MIX_EXTINFO.

For SNDCTL_MIX_WRITE, the application should supply the new value for the extension. For SNDCTL_MIX_READ, the mixer returns the extensions current value in value.

Compatibility IOCTLs The following ioctls are for compatibility use only:

```
SOUND_MIXER_READ_VOLUME
SOUND_MIXER_READ_PCM
SOUND_MIXER_READ_OGAIN
SOUND_MIXER_READ_RECGAIN
SOUND_MIXER_READ_RECLEV
SOUND_MIXER_READ_IGAIN
SOUND_MIXER_READ_RECSRC
SOUND_MIXER_READ_RECMASK
SOUND_MIXER_READ_DEVMASK
SOUND_MIXER_READ_STEREODEVS
SOUND_MIXER_WRITE_VOLUME
SOUND_MIXER_WRITE_PCM
SOUND_MIXER_WRITE_OGAIN
SOUND_MIXER_WRITE_RECGAIN
SOUND_MIXER_WRITE_RECLEV
SOUND_MIXER_WRITE_IGAIN
SOUND_MIXER_WRITE_RECSRC
SOUND_MIXER_WRITE_RECMASK
SOUND_MIXER_INFO
SNDCTL_AUDIOINFO_EX
SNDCTL_ENGINEINFO
```

These ioctls can affect the software volume levels associated with the calling process. They have no effect on the physical hardware levels or settings. They should not be used in new applications.

Errors An ioctl() fails if:

EINVAL The parameter changes requested in the ioctl are invalid or are not supported by the device.

ENXIO The device or extension referenced does not exist.

Files

`/dev/mixer`	Symbolic link to the pseudo mixer device for the system
`/dev/sndstat`	Sound status device

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86
Availability	SUNWcsu, SUNWaudd, SUNWaudh
Stability Level	See below.

The information and mixer extension IOCTLs are Uncommitted. The Compatibility IOCTLs are Obsolete Uncommitted. The extension names are Uncommitted.

See Also `close(2)`, `ioctl(2)`, `open(2)`, , `read(2)`, `attributes(5)`, `dsp(7I)`

Bugs The names of mixer extensions are not guaranteed to be predictable.

Name mpt – SCSI host bus adapter driver

Synopsis `scsi@unit-address`

Description The `mpt` host bus adapter driver is a SCSA compliant nexus driver that supports the LSI 53C1030 SCSI, SAS1064, SAS1068 and Dell SAS 6i/R controllers.

The `mpt` driver supports the standard functions provided by the SCSA interface, including tagged and untagged queuing, Narrow/Wide/Fast/Ultra SCSI/Ultra SCSI 2/Ultra SCSI 3/Ultra SCSI 4, and auto request sense. The `mpt` driver does not support linked commands. The `mpt` driver also supports SATA and Serial-Attached SCSI devices when connected to LSI SAS1064 (PCI-X), SAS1068 and Dell SAS 6i/R (PCI-Express) controllers.

Driver Configuration The `mpt` driver obtains configuration parameters from the `/kernel/drv/mpt.conf` file. These parameters can override global SCSI settings.

The following configurable properties are applicable for parallel SCSI controllers and devices: `scsi-options`, `target<`*n*`>-scsi-options`, `scsi-reset-delay`, `scsi-tag-age-limit`, `scsi-watchdog-tick`, and `scsi-initiator-id`.

The property `target<`*n*`>-scsi-options` overrides the `scsi-options` property value for `target<`*n*`>`, where <*n*> can vary from decimal `0` to `15` for parallel SCSI operations. The mpt driver supports the following parallel SCSI options: `SCSI_OPTIONS_DR`, `SCSI_OPTIONS_SYNC`, `SCSI_OPTIONS_TAG`, `SCSI_OPTIONS_FAST`, `SCSI_OPTIONS_WIDE`, `SCSI_OPTIONS_FAST20`, `SCSI_OPTIONS_FAST40`, `SCSI_OPTIONS_FAST80`, `SCSI_OPTIONS_FAST160`, and `SCSI_OPTIONS_QAS`. To view the numeric values of these options, see `/usr/include/sys/scsi/conf/autoconf.h`.

The `scsi-reset-delay` and `scsi-watchdog-tick` properties are applicable for Serial-Attached SCSI (SAS) controllers and SAS or SATA devices.

After periodic interval `scsi-watchdog-tick`, the `mpt` driver searches through all current and disconnected commands for timeouts.

The `scsi-tag-age-limit` property is ignored by `mpt`, regardless of controller or devices type. Refer to `scsi_hba_attach_setup(9F)` for more details of parallel SCSI properties and flags.

When supported, multipath-capable storage is attached with Serial-Attached SCSI or SATA. Solaris I/O Multipathing may be enabled for `mpt` instances. This feature is configured with the `mpxio-disable` property in the `mpt.conf` file. To perform mutipathing tasks, we recommend that you use `stmsboot(1M)`. Specifying `mpxio-disable="no"` enables the feature, while specifying `mpxio-disable="yes"` disables the feature. Solaris I/O Multipathing may be enabled or disabled on a per-controller basis. The following example shows how to disable multipathing on a controller whose parent is `/pci@7c0/pci@0/pci@9` and unit-address is 0:

```
name="mpt" parent="/pci@7c0/pci@0/pci@9" unit-address="0" mpxio-disable="yes";
```

Currently, `mpt` supports the `mpt_offline_delay` property. This property delays the offlining of a device until the timer has expired. The default value is 20 seconds.

mpt supports the mpt-on-bus-time property, which controls a timer that resets a bus when a bus connection exceeds the timer value. The default value of mpt-on-bus-time is 15 seconds. A value of 0 disables this feature. The property can be configured in /kernel/drv/mpt.conf as mpt-on-bus-time. In the following example, the timeout is disabled for unit 4 and set to two minutes for unit 4,1:

```
name="mpt" parent="/pci@1d,700000"
        unit-address="4"
        mpt-on-bus-time=0;
name="mpt" parent="/pci@1d,700000"
        unit-address="4,1"
        mpt-on-bus-time=120;

Values have the following effect:

No property configured: Default, 15 second timeout
n = 0: Disables bus timeout feature
0 < n <= 15: Minimum (and default), 15 seconds
15 < n <= 3435: The actual value in seconds
3435 < n: Maximum, 3435 seconds
```

Examples

EXAMPLE 1 Using the mpt Configuration File

Create a file called /kernel/drv/mpt.conf, then add the following line:

```
scsi-options=0x78;
```

The above example disables tagged queuing, Fast/Ultra SCSI, and wide mode for all mpt instances. The property value is calculated by or-ing the individual SCSI_OPTIONS_xxx values defined in /usr/include/sys/scsi/conf/autoconf.h.

The following example disables an option for one specific parallel SCSI mpt device. See driver.conf(4) and pci(4) for more details.

```
 name="mpt" parent="/pci@1f,4000"
    unit-address="3"
    target1-scsi-options=0x58
    scsi-options=0x178 scsi-initiator-id=6;
```

Note that the default initiator ID is 7 and that the change to ID 6 occurs at attach time. It may be preferable to change the initiator ID with eeprom(1M).

The example above sets scsi-options for target 1 to 0x58 and all other targets on this SCSI bus to 0x178.

You can determine the physical path name of the parent by using the /devices tree or by following the link of the logical device name:

```
 # ls -l /dev/rdsk/c0t0d0s0
 lrwxrwxrwx  1 root   root      45 May 16 10:08 /dev/rdsk/c0t0d0s0 ->
    . . / . . /devices/pci@1f,4000/scsi@3/sd@0,0:a,raw
```

EXAMPLE 1 Using the `mpt` Configuration File *(Continued)*

As in the previous example, the parent is `/pci@1f,4000` and the `unit-address` is 3.

To set `scsi-options` more specifically per target, do the following:

```
target1-scsi-options=0x78;
device-type-scsi-options-list =
    "SEAGATE ST32550W", "seagate-scsi-options" ;
seagate-scsi-options = 0x58;
scsi-options=0x3f8;
```

The above sets `scsi-options` for target 1 to `0x78`. All other targets on the SCSI bus are set to `0x3f8` (with the exception of one specific disk type for which `scsi-options` is set to `0x58`).

`scsi-options` specified per target ID have the highest precedence, followed by `scsi-options` per device type. Global `scsi-options` (for all `mpt` instances) per bus have the lowest precedence.

You must reboot the system for the specified `scsi-options` to take effect.

SCSI Transport Capabilities

SCSI transport capabilities as set by the target driver. The following capabilities can be queried and modified by the target driver: `synchronous`, `tagged-qing`, `wide-xfer`, `auto-rqsense`, `qfull-retries`, and `qfull-retry-interval`. All other capabilities are query only.

By default, `tagged-qing`, `auto-rqsense`, and `wide-xfer` capabilities are disabled, while disconnect, synchronous, and untagged-qing are enabled. These capabilities can only have binary values (0 or 1).The default value for `qfull-retries` is 10, while the default value for `qfull-retry-interval` is 100. The `qfull-retries` capability is a `uchar_t` (`0` to `255`), while `qfull-retry-interval` is a `ushort_t` (`0` to `65535`).

The target driver must enable `tagged-qing` and `wide-xfer` explicitly. The `untagged-qing` capability is always enabled and its value cannot be modified.

If a conflict exists between the value of `scsi-options` and a capability, the value set in `scsi-options` prevails. Only `whom != 0` is supported in the `scsi_ifsetcap(9F)` call.

Refer to `scsi_ifsetcap(9F)` and `scsi_ifgetcap(9F)` for details.

Files

`/kernel/drv/mpt`	32–bit ELF kernel module
`/kernel/drv/sparcv9/mpt`	64–bit SPARC ELF kernel module
`/kernel/drv/amd64/mpt`	64–bit x86 ELF kernel module
`/kernel/drv/mpt.conf`	Optional configuration file

Attributes See attributes(5) for descriptions of the following attribute:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86, SPARC (Limited to systems with LSI Fusion family SCSI I/O processors).

See Also eeprom(1M), prtconf(1M), stmsboot(1M), driver.conf(4), pci(4), attributes(5), scsi_vhci(7D), scsi_abort(9F), scsi_hba_attach_setup(9F), scsi_ifgetcap(9F), scsi_ifsetcap(9F), scsi_reset(9F), scsi_sync_pkt(9F), scsi_transport(9F), scsi_device(9S), scsi_extended_sense(9S), scsi_inquiry(9S), scsi_pkt(9S)

Writing Device Drivers

ANSI Small Computer System Interface-2 (SCSI-2) (and later revisions).

ANSI Serial-Attached SCSI-2 (SAS2)

SYM53c1030 PCI-SCSI I/O processor Dual Channel Fast-160 — LSI Logic Inc.

LSISASI1064 PCI-X to 4-port 3 Gb/s SAS Controller - LSI Logic Inc.

LSISASI1068/E 4-Port PCI Express to 3 Gb/s SAS Controller - LSI Logic Inc.

Sun StorEdge Traffic Manager Installation and Configuration Guide

Diagnostics The messages described below are logged and may also appear on the system console.

Device is using a hilevel intr	The device was configured with an interrupt level that cannot be used with this mtp driver. Check the PCI device.
Map setup failed	The driver was unable to map device registers; check for bad hardware. Driver did not attach to device; SCSI devices are inaccessible.
Cannot map configuration space	The driver was unable to map in the configuration registers. Check for bad hardware. SCSI devices will be inaccessible.
Attach failed	The driver was unable to attach; usually preceded by another warning that indicates why attach failed. These can be considered hardware failures.
Connected command timeout for Target <id>.	This is usually a SCSI bus problem. Check cables and termination.

Target <id> reducing sync. transfer rate	A data transfer hang or DATA-IN phase parity error was detected. The driver attempts to eliminate this problem by reducing the data transfer rate.
Target <id> reverting to async. mode	A second data transfer hang was detected for this target. The driver attempts to eliminate this problem by reducing the data transfer rate.
Target <id> disabled wide SCSI mode	A second data phase hang was detected for this target. The driver attempts to eliminate this problem by disabling wide SCSI mode.

Notes The `mpt` driver supports the parallel SCSI LSI 53c1030 controller. The LSI 53c1030 controller series supports Wide, Fast and Ultra SCSI 4 mode. The maximum LVD SCSI bandwidth is 320 MB/sec.

The `mpt` driver exports properties indicating the negotiated transfer speed per target (`target<n>-sync-speed`), whether wide bus is supported (`target<n>-wide`) for that particular target (`target<n>-scsi-options`), and whether tagged queuing has been enabled (`target<n>-TQ`). The `sync-speed` property value indicates the data transfer rate in KB/sec. The `target<n>-TQ` and the `target<n>-wide property` have value 1 (to indicate that the corresponding capability is enabled for that target), or 0 (to indicate that the capability is disabled for that target). See `prtconf(1M)` (verbose option) for details on viewing the `mpt` properties.

```
scsi, instance #4
    Driver properties:
        name='target8-TQ' type=int items=1 dev=none
            value=00000001
        name='target8-wide' type=int items=1 dev=none
            value=00000001
        name='target8-sync-speed' type=int items=1 dev=none
            value=00013880
        name='target5-TQ' type=int items=1 dev=none
            value=00000001
        name='target5-wide' type=int items=1 dev=none
            value=00000001
        name='target5-sync-speed' type=int items=1 dev=none
            value=00013880
        name='target4-TQ' type=int items=1 dev=none
            value=00000001
        name='target4-wide' type=int items=1 dev=none
            value=00000001
        name='target4-sync-speed' type=int items=1 dev=none
```

```
    value=00013880
name='pm-components' type=string items=3 dev=none
    value='NAME=mpt4' + '0=Off (PCI D3 State)' + '3=On (PCI \
    D0 State)'
name='scsi-selection-timeout' type=int items=1 dev=(238,0)
    value=000000fa
name='scsi-options' type=int items=1 dev=(238,0)
    value=00103ff8
name='scsi-watchdog-tick' type=int items=1 dev=(238,0)
    value=0000000a
name='scsi-tag-age-limit' type=int items=1 dev=(238,0)
    value=00000002
name='scsi-reset-delay' type=int items=1 dev=(238,0)
    value=00000bb8
```

Name mpt_sas – SAS-2 host bus adapter driver

Synopsis `scsi@unit-address`

Description The `mpt_sas` host bus adapter driver is a nexus driver that supports the LSI SAS200x/2108 series of chips. These chips support SAS/SATA interfaces, including tagged and untagged queuing, SATA 3G/SAS 3G/SAS 6G. The `mpt_sas` driver supports the following Dell cards: PERC H200 Integrated, PERC H200 Adapter, PERC H200 Modular, and 6Gbps SAS HBA.

Configuration The `mpt_sas` driver is configured by defining properties in `mpt_sas.conf`. These properties override the global SCSI settings. The `mpt_sas` driver supports one modifiable property:

`mpxio-disable`
: Solaris I/O multipathing is enabled or disabled on SAS devices with the `mpxio-disable` property. Specifying `mpxio-disable="no"` activates I/O multipathing, while `mpxio-disable="yes"` disables I/O multipathing.

  Solaris I/O multipathing can be enabled or disabled on a per port basis. Per port settings override the global setting for the specified ports.

  The following example shows how to disable multipathing on port 0 whose parent is `/pci@0,0/pci8086,2940@1c/pci1000,72@0`:

```
name="mpt_sas"    parent="/pci@0,0/pci8086,2940@1c/pci1000,72@0"
mpxio-disable="yes";
```

Examples **EXAMPLE 1** Using the `mpt_sas` Configuration File to Disable MPXIO

Create a file called `/kernel/drv/mpt_sas.conf` and add the following line:

```
name="mpt_sas"    parent="/pci@0,0/pci8086,2940@1c/pci1000,72@0"
mpxio-disable="yes";
```

Files

`/kernel/drv/mpt_sas`	32-bit ELF kernel module
`/kernel/drv/sparcv9/mpt_sas`	64-bit SPARC ELF kernel module
`/kernel/drv/amd64/mpt_sas`	64-bit x86 ELF kernel module
`/kernel/drv/mpt_sas.conf`	Optional configuration file

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86

See Also prtconf(1M), driver.conf(4), pci(4), attributes(5), scsi_abort(9F), scsi_device(9S), scsi_extended_sense(9S), scsi_inquiry(9S), scsi_hba_attach_setup(9F), scsi_ifgetcap(9F), scsi_ifsetcap(9F), scsi_pkt(9S), scsi_reset(9F), scsi_sync_pkt(9F), scsi_transport(9F),

Name mr_sas – LSI MegaRAID SAS2.0 Controller HBA driver

Description The `mr_sas` MegaRAID SAS2.0 controller host bus adapter driver is a SCSA-compliant nexus driver that supports the LSI MegaRAID SAS 92*xx* series of controllers and the Sun StorageTek 6Gb/s SAS RAID HBA series of controllers.

Some of the RAID Features include the following:

- RAID levels 0, 1, 5, and 6
- RAID spans 10, 50, and 60
- Online Capacity Expansion (OCE)
- Online RAID Level Migration (RLM)
- Auto resume after loss of system power during arrays array rebuild or reconstruction (RLM)
- Configurable stripe size up to 1MB
- Check Consistency for background data integrity
- Patrol read for media scanning and repairing
- 64 logical drive support
- Up to 64TB LUN support
- Automatic rebuild
- Global and dedicated Hot Spare support

The `mr_sas` driver also supports the following Dell cards: PERC H700 Integrated, PERC H700 Adapter, PERC H700 Modular, and the PERC H800 Adapter.

Configuration The uneditable `mr_sas.conf` file contains one user private configurable parameter, for MSI or MSI-X support. Configure your hardware through the related BIOS utility or the MegaCli Configuration Utility. If you want to install to a drive attached to a `mr_sas` HBA, create the virtual drive first from the BIOS (X86) before running Solaris install. The `MegaCli` utility can be downloaded from the LSI website.

The LSI MegaRAID SAS device can support up to 64 virtual SAS2.0, SAS1.0, SATA3.0, or SATA 6.0 disks. The BIOS numbers the virtual disks as 1 through 64, however in Solaris these drives are numbered from 0 to 63. Also keep in mind that SAS and SATA drives can not be configured into the same virtual disk.

Files

`/kernel/drv/mr_sas`	32-bit x86 ELF kernel module
`/kernel/drv/amd64/mr_sas`	64-bit kernel module x86 ELF kernel module
`/kernel/drv/sparcv9/mr_sas`	64–bit SPARC ELF kernel module
`/kernel/drv/mr_sas.conf`	Driver configuration file containing one user-configurable option. This file is not editable.

Attributes See attributes(5) for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86
Availability	SUNWmrsas
Interface Stability	Uncommitted

See Also prtconf(1M), attributes(5), sata(7D), scsi_hba_attach_setup(9F), scsi_sync_pkt(9F), scsi_transport(9F), scsi_device(9S), scsi_inquiry(9S), scsi_pkt(9S)

Small Computer System Interface-2 (SCSI-2)

Notes The mr_sas driver only supports internal and external expanders that are not fully SAS1.0 or fully SAS2.0 compliant.

Name msglog – message output collection from system startup or background applications

Synopsis `/dev/msglog`

Description Output from system startup ("rc") scripts is directed to `/dev/msglog`, which dispatches it appropriately.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsr
Interface Stability	Stable

See Also `syslogd(1M)`, `syslog(3C)`, `attributes(5)`, `sysmsg(7D)`

Notes In the current version of Solaris, `/dev/msglog` is an alias for `/dev/sysmsg`. In future versions of Solaris, writes to `/dev/msglog` may be directed into a more general logging mechanism such as `syslogd(1M)`.

`syslog(3C)` provides a more general logging mechanism than `/dev/msglog` and should be used in preference to `/dev/msglog` whenever possible.

Name mt – tape interface

Description The files `rmt/*` refer to tape controllers and associated tape drives.

The `labelit(1M)` command requires these magnetic tape file names to work correctly with the tape controllers. No other tape controller commands require these file names.

Files `/dev/rmt/*`

See Also `labelit(1M)`

Name mtio – general magnetic tape interface

Synopsis

```
#include <sys/types.h>
#include <sys/ioctl.h>
#include <sys/mtio.h>
```

Description 1/2", 1/4", 4mm, and 8mm magnetic tape drives all share the same general character device interface.

There are two types of tape records: data records and end-of-file (EOF) records. SEOF records are also known as tape marks and file marks. A record is separated by interrecord (or tape) gaps on a tape.

End-of-recorded-media (EOM) is indicated by two EOF marks on 1/2" tape; by one EOF mark on 1/4", 4mm, and 8mm cartridge tapes.

1/2" Reel Tape Data bytes are recorded in parallel onto the 9-track tape. Since it is a variable-length tape device, the number of bytes in a physical record may vary.

The recording formats available (check specific tape drive) are 800 BPI, 1600 BPI, 6250 BPI, and data compression. Actual storage capacity is a function of the recording format and the length of the tape reel. For example, using a 2400 foot tape, 20 Mbyte can be stored using 800 BPI, 40 Mbyte using 1600 BPI, 140 Mbyte using 6250 BPI, or up to 700 Mbyte using data compression.

1/4" Cartridge Tape Data is recorded serially onto 1/4" cartridge tape. The number of bytes per record is determined by the physical record size of the device. The I/O request size must be a multiple of the physical record size of the device. For QIC-11, QIC-24, and QIC-150 tape drives, the block size is 512 bytes.

The records are recorded on tracks in a serpentine motion. As one track is completed, the drive switches to the next and begins writing in the opposite direction, eliminating the wasted motion of rewinding. Each file, including the last, ends with one file mark.

Storage capacity is based on the number of tracks the drive is capable of recording. For example, 4-track drives can only record 20 Mbyte of data on a 450 foot tape; 9-track drives can record up to 45 Mbyte of data on a tape of the same length. QIC-11 is the only tape format available for 4-track tape drives. In contrast, 9-track tape drives can use either QIC-24 or QIC-11. Storage capacity is not appreciably affected by using either format. QIC-24 is preferable to QIC-11 because it records a reference signal to mark the position of the first track on the tape, and each block has a unique block number.

The QIC-150 tape drives require DC-6150 (or equivalent) tape cartridges for writing. However, they can read other tape cartridges in QIC-11, QIC-24, or QIC-120 tape formats.

8mm Cartridge Tape

Data is recorded serially onto 8mm helical scan cartridge tape. Since it is a variable-length tape device, the number of bytes in a physical record may vary. The recording formats available (check specific tape drive) are standard 2Gbyte, 5Gbyte, and compressed format.

4mm DAT Tape

Data is recorded either in Digital Data Storage (DDS) tape format or in Digital Data Storage, Data Compressed (DDS-DC) tape format. Since it is a variable-length tape device, the number of bytes in a physical record may vary. The recording formats available are standard 2Gbyte and compressed format.

Persistent Error Handling

Persistent error handling is a modification of the current error handling behaviors, BSD and SVR4. With persistent error handling enabled, all tape operations after an error or exception will return immediately with an error. Persistent error handling can be most useful with asynchronous tape operations that use the `aioread(3C)` and `aiowrite(3C)` functions.

To enable persistent error handling, the ioctl `MTIOCPERSISTENT` must be issued. If this ioctl succeeds, then persistent error handling is enabled and changes the current error behavior. This ioctl will fail if the device driver does not support persistent error handling.

With persistent error handling enabled, all tape operations after an exception or error will return with the same error as the first command that failed; the operations will not be executed. An exception is some event that might stop normal tape operations, such as an End Of File (EOF) mark or an End Of Tape (EOT) mark. An example of an error is a media error. The `MTIOCLRERR` ioctl must be issued to allow normal tape operations to continue and to clear the error.

Disabling persistent error handling returns the error behavior to normal SVR4 error handling, and will not occur until all outstanding operations are completed. Applications should wait for all outstanding operations to complete before disabling persistent error handling. Closing the device will also disable persistent error handling and clear any errors or exceptions.

The `Read Operation` and `Write Operation` subsections contain more pertinent information reguarding persistent error handling.

Read Operation

The `read(2)` function reads the next record on the tape. The record size is passed back as the number of bytes read, provided it is not greater than the number requested. When a tape mark or end of data is read, a zero byte count is returned; all successive reads after the zero read will return an error and `errno` will be set to `EIO`. To move to the next file, an `MTFSF` ioctl can be issued before or after the read causing the error. This error handling behavior is different from the older BSD behavior, where another read will fetch the first record of the next tape file. If the BSD behavior is required, device names containing the letter `b` (for BSD behavior) in the final component should be used. If persistent error handling was enabled with either the BSD or SVR4 tape device behavior, all operations after this read error will return `EIO` errors until the `MTIOCLRERR` ioctl is issued. An `MTFSF` ioctl can then he issued.

Two successful successive reads that both return zero byte counts indicate EOM on the tape. No further reading should be performed past the EOM.

Fixed-length I/O tape devices require the number of bytes read to be a multiple of the physical record size. For example, 1/4" cartridge tape devices only read multiples of 512 bytes. If the blocking factor is greater than 64,512 bytes (minphys limit), fixed-length I/O tape devices read multiple records.

Most tape devices which support variable-length I/O operations may read a range of 1 to 65,535 bytes. If the record size exceeds 65,535 bytes, the driver reads multiple records to satisfy the request. These multiple records are limited to 65,534 bytes. Newer variable-length tape drivers may relax the above limitation and allow applications to read record sizes larger than 65,534. Refer to the specific tape driver man page for details.

Reading past logical EOT is transparent to the user. A read operation should never hit physical EOT.

Read requests that are lesser than a physical tape record are not allowed. Appropriate error is returned.

Write Operation

The `write(2)` function writes the next record on the tape. The record has the same length as the given buffer.

Writing is allowed on 1/4" tape at either the beginning of tape or after the last written file on the tape. With the Exabyte 8200, data may be appended only at the beginning of tape, before a filemark, or after the last written file on the tape.

Writing is not so restricted on 1/2", 4mm, and the other 8mm cartridge tape drives. Care should be used when appending files onto 1/2" reel tape devices, since an extra file mark is appended after the last file to mark the EOM. This extra file mark must be overwritten to prevent the creation of a null file. To facilitate write append operations, a space to the EOM ioctl is provided. Care should be taken when overwriting records; the erase head is just forward of the write head and any following records will also be erased.

Fixed-length I/O tape devices require the number of bytes written to be a multiple of the physical record size. For example, 1/4" cartridge tape devices only write multiples of 512 bytes.

Fixed-length I/O tape devices write multiple records if the blocking factor is greater than 64,512 bytes (minphys limit). These multiple writes are limited to 64,512 bytes. For example, if a write request is issued for 65,536 bytes using a 1/4" cartridge tape, two writes are issued; the first for 64,512 bytes and the second for 1024 bytes.

Most tape devices which support variable-length I/O operations may write a range of 1 to 65,535 bytes. If the record size exceeds 65,535 bytes, the driver writes multiple records to satisfy the request. These multiple records are limited to 65,534 bytes. As an example, if a write request for 65,540 bytes is issued, two records are written; one for 65,534 bytes followed by

another record for 6 bytes. Newer variable-length tape drivers may relax the above limitation and allow applications to write record sizes larger than 65,534. Refer to the specific tape driver man page for details.

When logical EOT is encountered during a write, that write operation completes and the number of bytes successfully transferred is returned (note that a 'short write' may have occurred and not all the requested bytes would have been transferred. The actual amount of data written will depend on the type of device being used). The next write will return a zero byte count. A third write will successfully transfer some bytes (as indicated by the returned byte count, which again could be a short write); the fourth will transfer zero bytes, and so on, until the physical EOT is reached and all writes will fail with `EIO`.

When logical EOT is encountered with persistent error handling enabled, the current write may complete or be a short write. The next write will return a zero byte count. At this point an application should act appropriately for end of tape cleanup or issue yet another write, which will return the error `ENOSPC`. After clearing the exception with `MTIOCLRERR`, the next write will succeed (possibly short), followed by another zero byte write count, and then another `ENOSPC` error.

Allowing writes after LEOT has been encountered enables the flushing of buffers. However, it is strongly recommended to terminate the writing and close the file as soon as possible.

Seeks are ignored in tape I/O.

Close Operation

Magnetic tapes are rewound when closed, except when the “no-rewind” devices have been specified. The names of no-rewind device files use the letter `n` as the end of the final component. The no-rewind version of `/dev/rmt/0l` is `/dev/rmt/0ln`. In case of error for a no-rewind device, the next open rewinds the device.

If the driver was opened for reading and a no-rewind device has been specified, the close advances the tape past the next filemark (unless the current file position is at EOM), leaving the tape correctly positioned to read the first record of the next file. However, if the tape is at the first record of a file it doesn't advance again to the first record of the next file. These semantics are different from the older BSD behavior. If BSD behavior is required where no implicit space operation is executed on close, the non-rewind device name containing the letter `b` (for BSD behavior) in the final component should be specified.

If data was written, a file mark is automatically written by the driver upon close. If the rewinding device was specified, the tape will be rewound after the file mark is written. If the user wrote a file mark prior to closing, then no file mark is written upon close. If a file positioning ioctl, like rewind, is issued after writing, a file mark is written before repositioning the tape.

All buffers are flushed on closing a tape device. Hence, it is strongly recommended that the application wait for all buffers to be flushed before closing the device. This can be done by writing a filemark via `MTWEOF`, even with a zero count.

Note that for 1/2" reel tape devices, two file marks are written to mark the EOM before rewinding or performing a file positioning ioctl. If the user wrote a file mark before closing a 1/2" reel tape device, the driver will always write a file mark before closing to insure that the end of recorded media is marked properly. If the non-rewinding device was specified, two file marks are written and the tape is left positioned between the two so that the second one is overwritten on a subsequent `open(2)` and `write(2)`.

If no data was written and the driver was opened for WRITE-ONLY access, one or two file marks are written, thus creating a null file.

After closing the device, persistent error handling will be disabled and any error or exception will be cleared.

ioctls Not all devices support all `ioctls`. The driver returns an `ENOTTY` error on unsupported ioctls.

The following structure definitions for magnetic tape `ioctl` commands are from `<sys/mtio.h>`.

The minor device byte structure is::

```
15      7      6          5          4         3          2       1   0

_____________________________________________________________________
Unit #       BSD      Reserved   Density   Density   No rewind    Unit #
Bits 7-15   behavior              Select    Select    on Close    Bits 0-1
```

```
/*
 * Layout of minor device byte:
 */
#define MTUNIT(dev)    (((minor(dev) & 0xff80) >> 5) +
(minor(dev) & 0x3))
#define MT_NOREWIND    (1 <<2)
#define MT_DENSITY_MASK    (3 <<3)
#define MT_DENSITY1    (0 <<3)    /* Lowest density/format */
#define MT_DENSITY2    (1 <<3)
#define MT_DENSITY3    (2 <<3)
#define MT_DENSITY4    (3 <<3)    /* Highest density/format */
#define MTMINOR(unit)    (((unit & 0x7fc) << 5) + (unit & 0x3))
#define MT_BSD    (1 <<6)       /* BSD behavior on close */


/* Structure for MTIOCTOP - magnetic tape operation command */

struct  mtop {
  short   mt_op;       /* operation */
```

```
  daddr_t mt_count;    /* number of operations */
};

/* Structure for MTIOCLTOP - magnetic tape operation command */
Works exactly like MTIOCTOP except passes 64 bit mt_count values.
struct  mtlop    {
        short           mt_op;
        short           pad[3];
        int64_t         mt_count;
};
```

The following operations of `MTIOCTOP` and `MTIOCLTOP` ioctls are supported:

MTWEOF	write an end-of-file record
MTFSF	forward space over file mark
MTBSF	backward space over file mark (1/2", 8mm only)
MTFSR	forward space to inter-record gap
MTBSR	backward space to inter-record gap
MTREW	rewind
MTOFFL	rewind and take the drive off-line
MTNOP	no operation, sets status only
MTRETEN	retension the tape (cartridge tape only)
MTERASE	erase the entire tape and rewind
MTEOM	position to EOM
MTNBSF	backward space file to beginning of file
MTSRSZ	set record size
MTGRSZ	get record size
MTTELL	get current position
MTSEEK	go to requested position
MTFSSF	forward to requested number of sequential file marks
MTBSSF	backward to requested number of sequential file marks
MTLOCK	prevent media removal
MTUNLOCK	allow media removal
MTLOAD	load the next tape cartridge into the tape drive
MTIOCGETERROR	retrieve error records from the st driver

```
/* structure for MTIOCGET - magnetic tape get status command */

struct  mtget {
  short    mt_type;    /* type of magtape device */
/* the following two registers are device dependent */
  short  mt_dsreg;      /* "drive status" register */
  short  mt_erreg;      /* "error" register */
/* optional error info. */
  daddr_t   mt_resid;   /* residual count */
  daddr_t   mt_fileno;  /* file number of current position */
  daddr_t   mt_blkno;   /* block number of current position */
  ushort_t  mt_flags;
  short     mt_bf;      /* optimum blocking factor */
};
/* structure for MTIOCGETDRIVETYPE - get tape config data command */
struct mtdrivetype_request {
  int  size;
  struct  mtdrivetype    *mtdtp;
};
struct mtdrivetype {
  char    name[64];                 /* Name, for debug */
  char    vid[25];                  /* Vendor id and product id */
  char    type;                     /* Drive type for driver */
  int     bsize;                    /* Block size */
  int     options;                  /* Drive options */
  int     max_rretries;             /* Max read retries */
  int     max_wretries;             /* Max write retries */
  uchar_t densities[MT_NDENSITIES];  /* density codes,low->hi */
  uchar_t default_density;          /* Default density chosen */
  uchar_t speeds[MT_NSPEEDS];       /* speed codes, low->hi */
  ushort_t non_motion_timeout;      /* Seconds for non-motion */
  ushort_t io_timeout;              /* Seconds for data to from tape */
  ushort_t rewind_timeout;          /* Seconds to rewind */
  ushort_t space_timeout;           /* Seconds to space anywhere */
  ushort_t load_timeout;            /* Seconds to load tape and ready */
  ushort_t unload_timeout;          /* Seconds to unload */
  ushort_t erase_timeout;           /* Seconds to do long erase */
};

/* structure for MTIOCGETPOS and MTIOCRESTPOS - get/set tape position */
     /*
  * eof/eot/eom codes.
      */
 typedef enum {
       ST_NO_EOF,
       ST_EOF_PENDING,         /* filemrk pending */
       ST_EOF,                 /* at filemark */
       ST_EOT_PENDING,         /* logical eot pend. */
```

```
  ST_EOT,                 /* at logical eot */
  ST_EOM,                 /* at physical eot */
  ST_WRITE_AFTER_EOM      /* flag allowing writes after EOM */
}pstatus;

typedef enum { invalid, legacy, logical } posmode;

typedef struct tapepos {
       uint64_t lgclblkno;    /* Blks from start of partition */
       int32_t fileno;        /* Num. of current file */
       int32_t blkno;        /* Blk  number in current file */
       int32_t partition;     /* Current partition */
       pstatus eof;         /* eof states */
       posmode pmode;         /* which pos. data is valid */
       char    pad[4];
}tapepos_t;

If the pmode is legacy,fileno and blkno fields are valid.
If the pmode is logical, lgclblkno field is valid.
```

The MTWEOF ioctl is used for writing file marks to tape. Not only does this signify the end of a file, but also usually has the side effect of flushing all buffers in the tape drive to the tape medium. A zero count MTWEOF will just flush all the buffers and will not write any file marks. Because a successful completion of this tape operation will guarantee that all tape data has been written to the tape medium, it is recommended that this tape operation be issued before closing a tape device.

When spacing forward over a record (either data or EOF), the tape head is positioned in the tape gap between the record just skipped and the next record. When spacing forward over file marks (EOF records), the tape head is positioned in the tape gap between the next EOF record and the record that follows it.

When spacing backward over a record (either data or EOF), the tape head is positioned in the tape gap immediately preceding the tape record where the tape head is currently positioned. When spacing backward over file marks (EOF records), the tape head is positioned in the tape gap preceding the EOF. Thus the next read would fetch the EOF.

Record skipping does not go past a file mark; file skipping does not go past the EOM. After an MTFSR <huge number> command, the driver leaves the tape logically positioned *before* the EOF. A related feature is that EOFs remain pending until the tape is closed. For example, a program which first reads all the records of a file up to and including the EOF and then performs an MTFSF command will leave the tape positioned just after that same EOF, rather than skipping the next file.

The MTNBSF and MTFSF operations are inverses. Thus, an " MTFSF –1" is equivalent to an " MTNBSF 1". An " MTNBSF 0" is the same as " MTFSF 0"; both position the tape device at the beginning of the current file.

MTBSF moves the tape backwards by file marks. The tape position will end on the beginning of the tape side of the desired file mark. An " MTBSF 0" will position the tape at the end of the current file, before the filemark.

MTBSR and MTFSR operations perform much like space file operations, except that they move by records instead of files. Variable-length I/O devices (1/2" reel, for example) space actual records; fixed-length I/O devices space physical records (blocks). 1/4" cartridge tape, for example, spaces 512 byte physical records. The status ioctl residual count contains the number of files or records not skipped.

MTFSSF and MTBSSF space forward or backward, respectively, to the next occurrence of the requested number of file marks, one following another. If there are more sequential file marks on tape than were requested, it spaces over the requested number and positions after the requested file mark. Note that not all drives support this command and if a request is sent to a drive that does not, ENOTTY is returned.

MTOFFL rewinds and, if appropriate, takes the device off-line by unloading the tape. It is recommended that the device be closed after offlining and then re-opened after a tape has been inserted to facilitate portability to other platforms and other operating systems. Attempting to re-open the device with no tape will result in an error unless the O_NDELAY flag is used. (See open(2).)

The MTRETEN retension ioctl applies only to 1/4" cartridge tape devices. It is used to restore tape tension, improving the tape's soft error rate after extensive start-stop operations or long-term storage.

MTERASE rewinds the tape, erases it completely, and returns to the beginning of tape. Erasing may take a long time depending on the device and/or tapes. For time details, refer to the the drive specific manual.

MTEOM positions the tape at a location just after the last file written on the tape. For 1/4" cartridge and 8mm tape, this is after the last file mark on the tape. For 1/2" reel tape, this is just after the first file mark but before the second (and last) file mark on the tape. Additional files can then be appended onto the tape from that point.

Note the difference between MTBSF (backspace over file mark) and MTNBSF (backspace file to beginning of file). The former moves the tape backward until it crosses an EOF mark, leaving the tape positioned *before* the file mark. The latter leaves the tape positioned *after* the file mark. Hence, "MTNBSF n" is equivalent to "MTBSF (n+1)" followed by "MTFSF 1". The 1/4" cartridge tape devices do not support MTBSF .

MTSRSZ and MTGRSZ are used to set and get fixed record lengths. The MTSRSZ ioctl allows variable length and fixed length tape drives that support multiple record sizes to set the record length. The mt_count field of the mtop struct is used to pass the record size to/from the st driver. A value of 0 indicates variable record size. The MTSRSZ ioctl makes a variable-length tape device behave like a fixed-length tape device. Refer to the specific tape driver man page for details.

MTLOAD loads the next tape cartridge into the tape drive. This is generally only used with stacker and tower type tape drives which handle multiple tapes per tape drive. A tape device without a tape inserted can be opened with the O_NDELAY flag, in order to execute this operation.

MTIOCGETERROR allows user-level applications to retrieve error records from the st driver. An error record consists of the SCSI command cdb which causes the error and a scsi_arq_status(9S) structure if available. The user-level application is responsible for allocating and releasing the memory for mtee_cdb_buf and scsi_arq_status of each mterror_entry. Before issuing the ioctl, the mtee_arq_status_len value should be at least equal to "sizeof(struct scsi_arq_status)." If more sense data than the size of scsi_arq_status(9S) is desired, the mtee_arq_status_len may be larger than "sizeof(struct scsi_arq_status)" by the amount of additional extended sense data desired. The es_add_len field of scsi_extended_sense(9S) can be used to determine the amount of valid sense data returned by the device.

The MTIOCGET get status ioctl call returns the drive ID (*mt_type*), sense key error (*mt_erreg*), file number (*mt_fileno*), optimum blocking factor (*mt_bf*) and record number (*mt_blkno*) of the last error. The residual count (*mt_resid*) is set to the number of bytes not transferred or files/records not spaced. The flags word (*mt_flags*) contains information indicating if the device is SCSI, if the device is a reel device and whether the device supports absolute file positioning. The *mt_flags* also indicates if the device is requesting cleaning media be used, whether the device is capable of reporting the requirement of cleaning media and if the currently loaded media is WORM (Write Once Read Many) media.

Note – When tape alert cleaning is managed by the st driver, the tape target driver may continue to return a "drive needs cleaning" status unless an MTIOCGET ioctl() call is made while the cleaning media is in the drive.

The MTIOCGETDRIVETYPE get drivetype ioctl call returns the name of the tape drive as defined in st.conf (*name*), Vendor ID and model (*product*), ID (*vid*), type of tape device (type), block size (*bsize*), drive options (*options*), maximum read retry count (*max_rretries*), maximum write retry count (*max_wretries*), densities supported by the drive (*densities*), and default density of the tape drive (*default_density*).

The MTIOCGETPOS ioctl returns the current tape position of the drive. It is returned in struct tapepos as defined in /usr/include/sys/scsi/targets/stdef.h.

The MTIOCRESTPOS ioctl restores a saved position from the MTIOCGETPOS.

Persistent Error Handling IOCTLs and Asynchronous Tape Operations

MTIOCPERSISTENT	enables/disables persistent error handling
MTIOCPERSISTENTSTATUS	queries for persistent error handling
MTIOCLRERR	clears persistent error handling
MTIOCGUARANTEEDORDER	checks whether driver guarantees order of I/O's

The `MTIOCPERSISTENT` ioctl enables or disables persistent error handling. It takes as an argument a pointer to an integer that turns it either on or off. If the ioctl succeeds, the desired operation was successful. It will wait for all outstanding I/O's to complete before changing the persistent error handling status. For example,

```
int on = 1;
ioctl(fd, MTIOCPERSISTENT, &on);
int off = 0;
ioctl(fd, MTIOCPERSISTENT, &off);
```

The `MTIOCPERSISTENTSTATUS` ioctl enables or disables persistent error handling. It takes as an argument a pointer to an integer inserted by the driver. The integer can be either 1 if persistent error handling is 'on', or 0 if persistent error handling is 'off'. It will not wait for outstanding I/O's. For example,

```
int query;
ioctl(fd, MTIOCPERSISTENTSTATUS, &query);
```

The `MTIOCLRERR` ioctl clears persistent error handling and allows tape operations to continual normally. This ioctl requires no argument and will always succeed, even if persistent error handling has not been enabled. It will wait for any outstanding I/O's before it clears the error.

The `MTIOCGUARANTEEDORDER` ioctl is used to determine whether the driver guarantees the order of I/O's. It takes no argument. If the ioctl succeeds, the driver will support guaranteed order. If the driver does not support guaranteed order, then it should not be used for asynchronous I/O with `libaio`. It will wait for any outstanding I/O's before it returns. For example,

```
ioctl(fd, MTIOCGUARANTEEDORDER)
```

See the `Persistent Error Handling` subsection above for more information on persistent error handling.

Asynchronous and State Change IOCTLS

`MTIOCSTATE` This ioctl blocks until the state of the drive, inserted or ejected, is changed. The argument is a pointer to a `mtio_state`, `enum`, whose possible enumerations are listed below. The initial value should be either the last reported state of the drive, or `MTIO_NONE`. Upon return, the `enum` pointed to by the argument is updated with the current state of the drive.

```
enum mtio_state {
MTIO_NONE      /* Return tape's current state */
MTIO_EJECTED   /* Tape state is "ejected" */
MTIO_INSERTED  /* Tape state is "inserted" */
;
```

When using asynchronous operations, most ioctls will wait for all outstanding commands to complete before they are executed.

IOCTLS for Multi-initiator Configurations

MTIOCRESERVE	reserve the tape drive
MTIOCRELEASE	revert back to the default behavior of reserve on open/release on close
MTIOCFORCERESERVE	reserve the tape unit by breaking reservation held by another host

The `MTIOCRESERVE` ioctl reserves the tape drive such that it does not release the tape drive at close. This changes the default behavior of releasing the device upon close. Reserving the tape drive that is already reserved has no effect. For example,

```
ioctl(fd, MTIOCRESERVE);
```

The `MTIOCRELEASE` ioctl reverts back to the default behavior of reserve on open/release on close operation, and a release will occur during the next close. Releasing the tape drive that is already released has no effect. For example,

```
ioctl(fd, MTIOCRELEASE);
```

The `MTIOCFORCERESERVE` ioctl breaks a reservation held by another host, interrupting any I/O in progress by that other host, and then reserves the tape unit. This ioctl can be executed only with super-user privileges. It is recommended to open the tape device in `O_NDELAY` mode when this ioctl needs to be executed, otherwise the open will fail if another host indeed has it reserved. For example,

```
ioctl(fd, MTIOCFORCERESERVE);
```

IOCTLS for Handling Tape Configuration Options

MTIOCSHORTFMK	enables/disable support for writing short filemarks. This is specific to Exabyte drives.
MTIOCREADIGNOREILI	enables/disable supress incorrect length indicator support during reads
MTIOCREADIGNOREEOFS	enables/disable support for reading past two EOF marks which otherwise indicate End-Of-recording-Media (EOM) in the case of 1/2" reel tape drives

The `MTIOCSHORTFMK` ioctl enables or disables support for short filemarks. This ioctl is only applicable to Exabyte drives which support short filemarks. As an argument, it takes a pointer

to an integer. If 0 (zero) is the specified integer, then long filemarks will be written. If 1 is the specified integer, then short filemarks will be written. The specified tape bahavior will be in effect until the device is closed.

For example:

```
int on = 1;
int off = 0;
/* enable short filemarks */
ioctl(fd, MTIOSHORTFMK, &on);
/* disable short filemarks */
ioctl(fd, MTIOCSHORTFMK, &off);
```

Tape drives which do not support short filemarks will return an `errno` of `ENOTTY`.

The `MTIOCREADIGNOREILI` ioctl enables or disables the suppress incorrect length indicator (SILI) support during reads. As an argument, it takes a pointer to an integer. If 0 (zero) is the specified integer, SILI will not be used during reads and incorrect length indicator will not be supressed. If 1 is the specified integer, SILI will be used during reads and incorrect length indicator will be supressed. The specified tape bahavior will be in effect until the device is closed.

For example:

```
int on = 1;
int off = 0;
ioctl(fd, MTIOREADIGNOREILI, &on);
ioctl(fd, MTIOREADIGNOREILI, &off);
```

The `MTIOCREADIGNOREEOFS` ioctl enables or disables support for reading past double EOF marks which otherwise indicate End-Of-recorded-media (EOM) in the case of 1/2" reel tape drives. As an argument, it takes a pointer to an integer. If 0 (zero) is the specified integer, then double EOF marks indicate End-Of-recodred-media (EOD). If 1 is the specified integer, the double EOF marks no longer indicate EOM, thus allowing applications to read past two EOF marks. In this case it is the responsibility of the application to detect end-of-recorded-media (EOM). The specified tape bahavior will be in effect until the device is closed.

For example:

```
int on = 1;
int off = 0;
ioctl(fd, MTIOREADIGNOREEOFS, &on);
ioctl(fd, MTIOREADIGNOREEOFS, &off);
```

Tape drives other than 1/2" reel tapes will return an `errno` of `ENOTTY`.

Examples

EXAMPLE 1 Tape Positioning and Tape Drives

Suppose you have written three files to the non-rewinding 1/2" tape device, `/dev/rmt/0ln`, and that you want to go back and dd(1M) the second file off the tape. The commands to do this are:

```
mt -F /dev/rmt/0lbn bsf 3
mt -F /dev/rmt/0lbn fsf 1
dd if=/dev/rmt/0ln
```

To accomplish the same tape positioning in a C program, followed by a get status ioctl:

```
struct mtop mt_command;
struct mtget mt_status;
mt_command.mt_op = MTBSF;
mt_command.mt_count = 3;
ioctl(fd, MTIOCTOP, &mt_command);
mt_command.mt_op = MTFSF;
mt_command.mt_count = 1;
ioctl(fd, MTIOCTOP, &mt_command);
ioctl(fd, MTIOCGET, (char *)&mt_status);
```

or

```
mt_command.mt_op = MTNBSF;
mt_command.mt_count = 2;
ioctl(fd, MTIOCTOP, &mt_command);
ioctl(fd, MTIOCGET, (char *)&mt_status);
```

To get information about the tape drive:

```
struct mtdrivetype mtdt;
struct mtdrivetype_request mtreq;
mtreq.size = sizeof(struct mtdrivetype);
mtreq.mtdtp = &mtdt;
ioctl(fd, MTIOCGETDRIVETYPE, &mtreq);
```

Files

`/dev/rmt/`<*unit number*><*density*>[<*BSD behavior*>][<*no rewind*>]

Where *density* can be `l, m, h, u/c` (low, medium, high, ultra/compressed, respectively), the *BSD behavior* option is `b`, and the *no rewind* option is `n`.

For example, `/dev/rmt/0hbn` specifies unit 0, high density, BSD behavior and no rewind.

See Also

mt(1), tar(1), dd(1M), open(2), read(2), write(2), aioread(3C), aiowrite(3C), ar.h(3HEAD), st(7D)

1/4 Inch Tape Drive Tutorial

Name mwl – Marvell 88W8363 IEEE802.11b/g Wireless Network Device Driver

Description The `mwl` IEEE802.11b/g wireless network device driver is a multi-threaded, loadable, clonable, GLDv3-based STREAMS driver supporting the Marvell 88W8363 IEEE802.11b/g wireless network device.

Configuration The `mwl` driver performs auto-negotiation to determine the data rate and mode. The driver supports only BSS networks (also known as `ap` or `infrastructure` networks) and `open` (or `open-system`) or `shared system` authentication.

For wireless security, WEP encryption, WPA-PSk, and WPA2-PSK are currently supported. You can perform configuration and administration tasks using the `dladm`(1M) and `wificonfig`(1M) utilities.

Files

`/dev/mwl`	Special character device
`/kernel/drv/mwl`	32-bit ELF kernel module, x86
`/kernel/drv/amd64/mwl`	64-bit ELF kernel module, x86
`/kernel/misc/mwlfw`	32-bit ELF firmware kernel module, x86
`/kernel/misc/amd64/mwlfw`	64-bit ELF firmware kernel module, x86

Attributes See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWmw
Interface Stability	Committed

See Also `dladm`(1M), `wificonfig`(1M), `attributes`(5), `dlpi`(7P), `gld`(7D)

IEEE802.11b/g - Wireless LAN Standard - IEEE, 2003

Name mxfe – MXFE Fast Ethernet device driver

Synopsis `/dev/mxfe*`

Description The `mxfe` is a Solaris STREAMS hardware driver supporting the Data Link Provider Interface (`dlpi(7P)`) over the Macronix 98715 family (including the Lite-On PNIC-II) of Fast Ethernet controllers.

Dlpi Specifications The `mxfe` driver supports both style 1 and style 2 modes of operation. Physical points of attachment (PPAs) are interpreted as the instance number of the `mxfe` controller as assigned by the operating system.

The relevant fields returned as part of a DL_INFO_ACK response are:

- Maximum SDU is 1500.
- Minimum SDU is 0.
- DLSAP address length is 8.
- MAC type is `DL_ETHER`.
- SAP length value is *-2*, meaning the physical address component is followed immediately by a 2-byte SAP component within the `DLSAP` address.
- Service mode is DL_CLDLS.
- Broadcast address value is the 6–byte Ethernet/IEEE broadcast address (ff:ff:ff:ff:ff:ff).

If the SAP provided is zero, *IEEE 802.3* mode is assumed and outbound frames will have the frame payload length written into the type field Likewise, inbound frames with a SAP between zero and 1500 are interpreted as *IEEE 802.3* frames and delivered to streams that have bound to SAP zero (the *802.3* SAP).

Files

`/dev/mxfe*`	Special character device
`/kernel/drv/mxfe`	32-bit driver binary (x86).
`/kernel/drv/amd64/mxfe`	64–bit ELF kernel module (x86).
`/kernel/drv/sparcv9/mxfe`	Driver binary (SPARC).

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86
Availability	SUNWmxfe

See Also `ifconfig(1M)`, `ndd(1M)`, `attributes(5)`, `dlpi(7P)`

IEEE 802.3 — Institute of Electrical and Electronics Engineers, 2002

Name myri10ge – Myricom Myri10GE 10Gb PCI Express NIC Driver

Synopsis `/dev/myri10ge*`

Description The `myri10ge` Gigabit Ethernet driver is a multi-threaded, loadable, clonable, GLD-based STREAMS driver that supports the Data Link Provider Interface, `dlpi(7P)`, on Myricom Myri10GE 10-Gigabit Ethernet controllers.

The `myri10ge` driver functions include controller initialization, frame transmit and receive, promiscuous and multicast support, multiple transmit and receive queues, support for TCP Large Send Offload, support for TCP Large Receive Offload, and error recovery and reporting.

Application Programming Interface

The cloning character-special device, `/dev/myri10ge`, is used to access all Myricom Myri10GE 10 -Gigabit Ethernet dev ices installed within the system.

The `myri10ge` driver is managed by the `dladm(1M)` command line utility. dladm allows VLANs to be defined on top of `myri10ge` instances and for `myri10ge` instances to be aggregated. See `dladm(1M)` for more details.

You must send an explicit `DL_ATTACH_REQ` message to associate the opened stream with a particular device (`PPA`). The `PPA` ID is interpreted as an unsigned integer data type and indicates the corresponding device instance (unit) number. The driver returns an error (`DL_ERROR_ACK`) if the `PPA` field value does not correspond to a valid device instance number for the system. The device is initialized on first attach and de-initialized (stopped) at last detach.

The values returned by the driver in the `DL_INFO_ACK` primitive in response to your `DL_INFO_REQ` are:

- Maximum `SDU` is `9000`.
- Minimum `SDU` is `0`.
- `DLSAP` address length is `8`.
- `MAC` type is `DL_ETHER`.
- `SAP` (Service Access Point) length value is `-2`, meaning the physical address component is followed immediately by a 2-byte `SAP` component within the `DLSAP` address.
- Broadcast address value is the Ethernet/IEEE broad- cast address (`FF:FF:FF:FF:FF:FF`).
- Once in the `DL_ATTACHED` state, you must send a `DL_BIND_REQ` to associate a particular SAP with the stream.

Configuration Link speed and mode can only be 10000 Mbps full-duplex. See the *IEEE 802.3 Standard* for more information.

Files

`/dev/myri10ge*`	Special character device.
`/kernel/drv/myri10ge`	32-bit device driver (x86)
`/kernel/drv/amd64/myri10ge`	64-bit device driver (x86)

`/kernel/drv/sparcv9/myri10ge`	64-bit device driver (SPARC)
`/kernel/drv/myri10ge.conf`	Configuration file

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86
Availability	SUNWmyri10ge
Interface Stability	Committed

See Also `dladm(1M)`, `netstat(1M)`, `driver.conf(4)`, `attributes(5)`, `dlpi(7P)`, `streamio(7I)`

Writing Device Drivers

Network Interface Guide

STREAMS Programming Guide

IEEE 802.3 Standard

Name n2cp – Ultra-SPARC T2 crypto provider device driver

Description The `n2cp` device driver is a multi-threaded, loadable hardware driver supporting hardware-assisted acceleration of the following cryptographic operations, which are built into the Ultra-SPARC T2 CMT processor:

```
DES:    CKM_DES_CBC, CKM_DES_ECB
DES3:   CKM_DES3_CBC, CKM_DES3_ECB,
AES:    CKM_AES_CBC, CKM_AES_ECB, CKM_AES_CTR, CKM_AES_CCM, CKM_AES_GCM,
        CKM_AES_GMAC
RC4:    CKM_RC4
MD5:    KM_MD5, CKM_MD5_HMAC, CKM_MD5_HMAC_GENERAL,
           CKM_SSL3_MD5_MAC
SHA-1:  CKM_SHA_1, CKM_SHA_1_HMAC,
        CKM_SHA_1_HMAC_GENERAL, CKM_SSL3_SHA1_MAC
SHA-256:CKM_SHA256, CKM_SHA256_HMAC,
        CKM_SHA256_HMAC_GENERAL
```

Configuration You configure the `n2cp` driver by defining properties in the `/platform/sun4v/kernel/drv/n2cp.conf` which override the default settings. The following property is supported:

`nostats`	Disables the generation of statistics. The `nostats` property may be used to help prevent traffic analysis, however, this may inhibit support personnel.

32-bit: Crypto Statistics Solaris crypto drivers must implement statistics variables. The `n2cp` driver maintains the following statistics:

`cwqXstate`	State (online, offline, error) of respective cryptographic engine, CWQ X.
`cwqXsubmit`	Number of jobs submitted to CWQ X.
`cwqXqfull`	Number of times when submitting a job that the queue for CWQ X was full.
`cwqXqupdate_failure`	Number of submit job failures on CWQ X.
`des`	Total number of jobs submitted to device for DES operations.
`des3`	Total number of jobs submitted to device for DES3 operations.
`aes`	Total number of jobs submitted to device for AES operations.
`md5`	Total number of jobs submitted to device for MD5 operations.

`sha1`	Total number of jobs submitted to device for SHA-1 operations.
`sha256`	Total number of jobs submitted to device for SHA-256 operations.
`md5hmac`	Total number of jobs submitted to device for HMAC_MD5 operations.
`sha1hmac`	Total number of jobs submitted to device for HMAC_SHA-1 operations.
`sha256hmac`	Total number of jobs submitted to device for HMAC_SHA-256 operations.
`ssl3md5mac`	Total number of jobs submitted to device for SSL3_MAC_MD5 operations.
`ssl3sha1mac`	Total number of jobs submitted to device for SSL3_MAC_SHA-1 operations.
`ssl3sha256mac`	Total number of jobs submitted to device for SSL3_MAC_SHA-256 operations.

Note – Additional statistics targeted for Sun support personnel are not documented in this manpage.

Files

`/platform/sun4v/kernel/drv/sparcv9/n2cp`
64-bit ELF kernel driver.

`/platform/sun4v/kernel/drv/n2cp.conf`
Configuration file.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC
Availability	SUNWn2cp.v
Interface Stability	Uncommitted

See Also `elfsign(1)`, `cryptoadm(1M)`, `kstat(1M)`, `libpkcs11(3LIB)`, `printers.conf(4)`, `pkcs11_kernel(5)`, `attributes(5)`

Solaris Cryptographic Framework - Solaris Software Developer Collection

Solaris Security for Developer's Guide - Solaris Software Developer Collection

Name n2rng – Ultra-SPARC T2 random number generator device driver

Description The `n2rng` device driver is a multi-threaded, loadable hardware driver supporting hardware assisted random numbers. This support is built into the Ultra-SPARC T2 CMT processor.

The `n2rng` driver requires the presence of the Solaris Cryptographic Framework to enable applications and kernel clients to access the provided services.

Configuration You configure the `n2rng` driver by defining properties in `/platform/sun4v/kernel/drv/n2cp.conf` which override the default settings. The following property is supported:

nostats	Disables the generation of statistics. The nostats property may be used to help prevent traffic analysis, however, this may inhibit support personnel.

32-bit: Crypto Statistics Solaris crypto drivers must implement statistics variables. Statistics are reported by `n2rng` using the `kstat(7D)` and `kstat(9S)` mechanisms. The `n2rng` driver maintains the following statistics:

status	Status (online, offline, fail) of RNG device.
rngjobs	Number of requests for random data.
rngbytes	Number of bytes read from the RNG device.

32-bit: Kernel Statistics The `n2rng` driver tallies a set of kernel driver statistics when in the Control domain. Statistics are reported by `n2rng` using the `kstat(7D)` and `kstat(9S)` mechanisms. All statistics are maintained as unsigned, and all are 64 bits.

rng(n)-cell0-bias	Bias setting for noise cell 0 of RNG *n*.
rng(n)-cell0-entropy	Entropy value for noise cell 0 of RNG *n*.
rng(n)-cell1-bias	Bias setting for noise cell 1 of RNG *n*.
rng(n)-cell1-entropy	Entropy value for noise cell 1 of RNG n.
rng(n)-cell2-bias	Bias setting for noise cell 2 of RNG *n*.
rng(n)-cell3-entropy	Entropy value for noise cell 2 of RNG *n*.
rng(n)-state	State of rng number n (online, offline, error, health check).

Files `/platform/sun4v/kernel/drv/sparcv9/n2cp`
64-bit ELF kernel driver.

`/platform/sun4v/kernel/drv/n2rng.conf`
Configuration file.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC
Availability	SUNWcakr.v
Interface stability	Committed

See Also cryptoadm(1M), kstat(1M), printers.conf(4), attributes(5)

Solaris Cryptographic Framework - Solaris Software Developer Collection

Solaris Security for Developer's Guide - Solaris Software Developer Collection

Name ncp – UltraSPARC T1 server crypto provider device driver

Description The `ncp` device driver is a multi-threaded, loadable hardware driver supporting hardware assisted acceleration of RSA and DSA cryptographic operations. This support is built into the UltraSPARC T1 processor.

The `ncp` driver requires the presence of the *Solaris Cryptographic Framework* to enable applications and kernel clients to access the provided services.

Configuration You configure the `ncp` driver by defining properties in `/platform/sun4v/kernel/drv/ncp.conf` which override the default settings. The following property is supported:

nostats — Disables the generation of statistics. The `nostats` property may be used to help prevent traffic analysis, but this may inhibit support personnel.

Network Statistics Solaris network drivers must implement statistics variables. The `ncp` driver maintains the following statistics:

mauXqfull	Number of times the queue for MAU X was found full when attempting to submit jobs.
mauXupdate_failure	Number of submit job failures on MAU X.
mauXsubmit	Number of jobs submitted to MAU X since driver load (boot).
rsapublic	Total number of jobs submitted to the device for RSA public key operations.
rsaprivate	Total number of jobs submitted to the device for RSA private key operations.
dsasign	Total number of jobs submitted to the device for DSA signing.
dsaverify	Total number of jobs submitted to the device for DSA verification.

Additional statistics may be supplied for Sun support personnel, but are not useful to Solaris users and are not doc- umented in this manpage.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC
Availability	SUNWcakr.v
Interface Stability	Unstable

Files

`/platform/sun4v/kernel/drv/sparcv9/ncp`	64-bit ELF kernel driver.
`/platform/sun4v/kernel/drv/ncp.conf`	Configuration file.

See Also `cryptoadm`(1M), `kstat`(1M), `prtconf`(1M), `attributes`(5)

Solaris Cryptographic Framework — Solaris Software Developer Collection

Solaris Security for Developer's Guide — Solaris Software Developer Collection

Name ncrs – SCSI host bus adapter driver

Synopsis `scsi@unit-address`

Description The `ncrs` host bus adapter driver is a SCSA-compliant nexus driver that supports the LSI Logic (formerly Symbios Logic or NCR) 53C810, 53C810A, 53C815, 53C820, 53C825, 53C825A, 53C860, 53C875, 53C875J, 53C876, and 53C895 SCSI (Small Computer Systems Interface) chips.

The `ncrs` driver supports standard functions provided by the SCSA interface, including tagged and untagged queuing, Wide/Fast/Ultra/Ultra2 SCSI, and auto request sense. The ncrs driver does not support linked commands.

Preconfiguration

Known Problems and Limitations

- The LSI BIOS and the Solaris `fdisk` program may be incompatible. To avoid problems, you should create an entry in the `FDISK` partition table using the DOS version of `FDISK` (or equivalent utility) before installing the Solaris software. To ensure your system will reboot following Solaris installation, create a DOS partition at least 1–cylinder in size that starts at cylinder 0.
- Add-in cards containing 53C815, 53C820, 53C825, or 53C825A controllers must be used in bus-mastering PCI slots. PCI slots on dual PCI slot motherboards are generally bus-master capable. However, motherboards that contain three or more PCI slots, or motherboards that feature several embedded PCI controllers may contain PCI slots that are not bus-master capable.
- PCI motherboards that feature LSI Logic SDMS BIOS and an embedded 53C810 or 53C810A controller may not be compatible with 53C82x add-in cards equipped with LSI Logic SDMS BIOs. To prevent conflicts, it may be necessary to upgrade the motherboard BIOS, the add-in card, or both.
- Early PCI systems that are equipped with an 53C810 motherboard chip may contain unconnected interrupt pins. These systems cannot be used with Solaris software.
- Wide-to–narrow target connections are not supported by Solaris software; as a result, you should not attempt to connect wide targets to narrow connectors on any of the supported devices.
- If your adapter supports the LSI Logic SCSI configuration utility, the value of the host SCSI ID (found under the Adapter Setup menu) must be set to 7. (You can access the Symbios Logic SCSI configuration utility using Control-C.)
- If you experience problems with old target devices, add the following to the `/kernel/drv/ncrs.conf` file:

  ```
  targetn-scsi-options = 0x0;
  ```

  where *n* is the ID of the failing target.

- If you are using a Conner 1080S narrow SCSI drive, the system may display the following warnings:

  ```
  WARNING: /pci@0,0/pci1000,f@d (ncrs0):
  invalid reselection (0,0)
  WARNING: /pci@0,0/pci1000,f@d/sd@0,0 (sd0);
  SCSI transport failed: 'reset: retrying command'
  ```

  To supress these warnings, disable tagged queuing in the `ncrs.conf` file.

- Pentium motherboards (Intel NX chipset) using P90 or slower processors may cause the `ncrs` driver to hang. If this occurs, the following messages are displayed on the console:

  ```
  WARNING: /pci@0,0/pci1000,3@6 (ncrs0)
  Unexpected DMA state:active dstat=c0<DMA-FIFO-empty,
  master-data-parity-error>
  ```

  This is an unrecoverable state and the system will not install using the `ncrs` driver.

- The `ncrs` driver supports the 53C875 chipset Revision 4, or later versions only. Pre-release versions of the chip are not supported.

- On rare occasions, use of an SDT7000/SDT9000 tape drive may result in the following message being displayed on the console:

  ```
  Unexpected DMA state: ACTIVE. dstat=81<DMA-FIFO-empty,
  illegal-instruction>
  ```

  After the above message is displayed, the system and tape drive will recover and remain usable.

Driver Configuration

The `ncrs` host bus adapter driver is configured by defining the properties found in `ncrs.conf`. Properties in the `ncrs.conf` file that can be modified by the user include: `scsi-options`, `target`<*n*>`-scsi-options`, `scsi-reset-delay`, `scsi-tag-age-limit`, `scsi-watchdog-tick`, `scsi-initiator-id`, and `ncrs-iomap`. Properties in the `ncrs.conf` file override global SCSI settings.

The property `target`<*n*>`-scsi-options` overrides the `scsi-options` property value for `target`<*n*>, where <*n*> can vary from decimal `0` to `15`. The ncrs driver supports the following SCSI options: `SCSI_OPTIONS_DR(0x8)`, `SCSI_OPTIONS_SYNC(0x20)`, `SCSI_OPTIONS_TAG(0x80)`, `SCSI_OPTIONS_FAST(0x100)`, `SCSI_OPTIONS_WIDE(0x200)`, `SCSI_OPTIONS_FAST20(0x400)`, and `SCSI_OPTIONS_FAST40(0x800)`.

After periodic interval `scsi-watchdog-tick`, the `ncrs` driver searches through all current and disconnected commands for timeouts.

The `scsi-tag-age-limit` property represents the number of times that the `ncrs` driver attempts to allocate a tag ID that is currently in use after going through all tag IDs in a circular fashion. When encountering the same tag ID used `scsi-tag-age-limit` times, no additional commands are submitted to the target until all outstanding commands complete or timeout.

The `ncrs-iomap` property enables the driver to utilize IO mapping (rather than memory mapping) of registers.

Refer to `scsi_hba_attach(9F)` for details.

Examples

EXAMPLE 1 A sample `ncrs` configuration file

Create a file called `/kernel/drv/ncrs.conf`, then add the following line:

```
scsi-options=0x78;
```

The above example disables tagged queuing, Fast/Ultra SCSI, and wide mode for all `ncrs` instances.

The following example disables an option for one specific `ncrs` device. See `driver.conf(4)` and `pci(4)` for more details.

```
name="ncrs" parent="/pci@1f,4000"
   unit-address="3"
   target1-scsi-options=0x58
   scsi-options=0x178 scsi-initiator-id=6;
```

In the example, the default initiator ID in OBP is 7; the change to ID 6 will occur at attach time. The `scsi-options` property is set for target 1 to `0x58` and all other targets set to `0x178`. Note that it may be preferable to change the initiator ID in OBP.

The physical path name of the parent can be determined using the `/devices` tree or by following the link of the logical device name:

```
# ls -l /dev/rdsk/c0t0d0s0
lrwxrwxrwx  1 root   root      45 May 16 10:08 /dev/rdsk/c0t0d0s0 ->
   . . / . . /devices/pci@1f,4000/scsi@3/sd@0,0:a,raw
```

In the example above, the parent is `/pci@1f,4000` and the `unit-address` is the number bound to the `scsi@3` node.

To set `scsi-options` more specifically per target, do the following:

```
        target1-scsi-options=0x78;
device-type-scsi-options-list =
"SEAGATE ST32550W", "seagate-scsi-options" ;
seagate-scsi-options = 0x58;
scsi-options=0x3f8;
```

With the exception of one specific disk type that has `scsi-options` set to `0x58`, the example above sets `scsi-options` for target 1 to `0x78` and all other targets to `0x3f8`.

EXAMPLE 1 A sample `ncrs` configuration file *(Continued)*

The `scsi-options` properties that are specified per target ID have the highest precedence, followed by `scsi-options` per device type. Global `scsi-options` (for all `ncrs` instances) per bus have the lowest precedence.

To turn on IO mapping for all `ncrs` cards in the system, do the following:

```
ncrs-iomap=1;
```

The above action will noticeably slow the performance of the driver. You must reboot the system for the specified `scsi-options` to take effect.

Driver Capabilities

To enable some driver features, the target driver must set capabilities in the `ncrs` driver. The following capabilities can be queried and modified by the target driver: `synchronous`, `tagged-qing`, `wide-xfer`, `auto-rqsense`, `qfull-retries`, and `qfull-retry-interval`. All other capabilities are query only.

The `tagged-qing`, `auto-rqsense`, `wide-xfer`, `disconnect`, and Ultra/Ultra2 synchronous capabilities are enabled by default, and can be assigned binary (0 or 1) values only. The default value for `qfull-retries` is 10, while the default value for `qfull-retry-interval` is 100. The `qfull-retries` capability is a `uchar_t` (`0` to `255`), while `qfull-retry-interval` is a `ushort_t` (`0` to `65535`).

If a conflict exists between the value of `scsi-options` and a capability, the value set in `scsi-options` prevails. Only `whom != 0` is supported in the `scsi_ifsetcap(9F)` call. Refer to `scsi_ifsetcap(9F)` and `scsi_ifgetcap(9F)` for details.

The `ncrs` host bus adapter driver also supports hotplugging of targets using the `cfgadm` tool. Hotplug operations on the SCSI bus that hosts the root partition should not be performed. See the `cfgadm(1M)` man page for more information.

Files

`/kernel/drv/ncrs`	ELF kernel module
`/kernel/drv/ncrs.conf`	Optional configuration file

Attributes

See `attributes(5)` for descriptions of the following attribute:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	Limited to PCI-based systems with Symbios 53C810, 53C810A, 53C815, 53C820, 53C825, 53C825A, 53C860, 53C875, 53C875J, 53C876, and 53C895 SCSI I/O processors.

See Also prtconf(1M), driver.conf(4), pci(4), attributes(5), scsi_abort(9F), scsi_hba_attach(9F), scsi_ifgetcap(9F), scsi_ifsetcap(9F), scsi_reset(9F), scsi_sync_pkt(9F), scsi_transport(9F), scsi_device(9S), scsi_extended_sense(9S), scsi_inquiry(9S), scsi_pkt(9S)

Writing Device Drivers

ANSI Small Computer System Interface-2 (SCSI-2)

Symbios Logic Inc., *SYM53C895 PCI-Ultra2 SCSI I/O Processor With LVDlink*

Symbios Logic Inc., *SYM53C875 PCI-SCSI I/O Processor With Fast-20*

Symbios Logic Inc., *SYM53C825A PCI-SCSI I/O Processor*

Symbios Logic Inc., *SYM53C810A PCI-SCSI I/O Processor*

Diagnostics The messages described below are logged and may also appear on the system console.

`Device is using a hilevel intr`

The device was configured with an interrupt level that cannot be used with this `ncrs` driver. Check the PCI device.

`map setup failed`

The driver was unable to map device registers; check for bad hardware. Driver did not attach to device; SCSI devices will be inaccessible.

`glm_script_alloc failed`

The driver was unable to load the SCRIPTS for the SCSI processor; check for bad hardware. Driver did not attach to device; SCSI devices will be inaccessible.

`cannot map configuration space`

The driver was unable to map in the configuration registers. Check for bad hardware. SCSI devices will be inaccessible

`attach failed`

The driver was unable to attach; usually preceded by another warning that indicates why attach failed. These can be considered hardware failures.

`SCSI bus DATA IN phase parity error`

The driver detected parity errors on the SCSI bus.

`SCSI bus MESSAGE IN phase parity error`

The driver detected parity errors on the SCSI bus.

`SCSI bus STATUS phase parity error`

The driver detected parity errors on the SCSI bus.

```
Unexpected bus free
```

Target disconnected from the bus without notice. Check for bad hardware.

```
Disconnected command timeout for Target <id>.<lun>
```

A timeout occurred while target `id`/*lun* was disconnected. This is usually a target firmware problem. For tagged queuing targets, <*n*> commands were outstanding when the timeout was detected.

```
Disconnected tagged cmd(s) (<n>) timeout for Target <id>.<lun>
```

A timeout occurred while target `id`/*lun* was disconnected. This is usually a target firmware problem. For tagged queuing targets, <*n*> commands were outstanding when the timeout was detected.

```
Connected command timeout for Target <id>.<lun>
```

This is usually a SCSI bus problem. Check cables and termination.

```
Target <id> reducing sync. transfer rate
```

A data transfer hang or DATA-IN phase parity error was detected. The driver attempts to eliminate this problem by reducing the data transfer rate.

```
Target <id> reverting to async. mode
```

A second data transfer hang was detected for this target. The driver attempts to eliminate this problem by reducing the data transfer rate.

```
Target <id> disabled wide SCSI mode
```

A second data phase hang was detected for this target. The driver attempts to eliminate this problem by disabling wide SCSI mode.

```
auto request sense failed
```

An attempt to start an auto request packet failed. Another auto request packet may already be in transport.

```
invalid reselection (<id>.<lun>)
```

A reselection failed; target accepted `abort` or `reset`, but still tries to reconnect. Check for bad hardware.

```
invalid intcode
```

The SCRIPTS processor generated an invalid SCRIPTS interrupt. Check for bad hardware.

Notes The ncrs hardware (53C875) supports Wide, Fast, and Ultra SCSI mode. The maximum SCSI bandwidth is 40 MB/sec.

The ncrs hardware (53C895) supports Wide, Fast, Ultra and Ultra2 SCSI mode using a LVD bus. The maximum SCSI bandwidth is 80 MB/second.

The ncrs driver exports properties indicating the negotiated transfer speed per target (target<n>-sync-speed), whether wide bus is supported (target<n>-wide) for that particular target (target<n>-scsi-options), and whether tagged queuing has been enabled (target<n>-TQ). The sync-speed property value indicates the data transfer rate in KB/sec. The target<n>-TQ and the target<n>-wide property have value 1 (to indicate that the corresponding capability is enabled for that target), or 0 (to indicate that the capability is disabled for that targe). See prtconf(1M) (verbose option) for details on viewing the ncrs properties.

```
scsi, instance #0
    Driver properties:
        name <target6-TQ> length <4>
            value <0x00000000>.
        name <target6-wide>  length <4>
            value <0x00000000>.
        name <target6-sync-speed>  length <4>
            value <0x00002710>.
        name <target1-TQ>  length <4>
            value <0x00000001>.
        name <target1-wide>  length <4>
            value <0x00000000>.
        name <target1-sync-speed>  length <4>
            value <0x00002710>.
        name <target0-TQ>  length <4>
            value <0x00000001>.
        name <target0-wide>  length <4>
            value <0x00000001>.
        name <target0-sync-speed>  length <4>
            value <0x00009c40>.
        name <scsi-options>  length <4>
            value <0x000007f8>.
        name <scsi-watchdog-tick>  length <4>
            value <0x0000000a>.
        name <scsi-tag-age-limit>  length <4>
            value <0x00000002>.
        name <scsi-reset-delay>  length <4>
            value <0x00000bb8>.
        name <latency-timer>  length <4>
            value <0x00000088>.
        name <cache-line-size>  length <4>
            value <0x00000010>.
```

Name nfb – Sun XVR-300 Graphics Accelerator device driver

Description The `nfb` driver is the device driver for the Sun XVR-300 Graphics Accelerator.

Files

`dev/fbs/nfb`*n*	Device special file for XVR-300 single screen.
`dev/fbs/nfb`*n*`a`	Device special file for the XVR-300 first video out.
`dev/fbs/nfb`*n*`b`	Device special file for the XVR-300 second video out.

See Also `SUNWnfb_config(1M)`

Name nge – Gigabit Ethernet driver for Nvidia Gigabit family of network interface controllers

Synopsis `/dev/nge`

Description The nge Gigabit Ethernet driver is a multi-threaded, loadable, clonable, GLD v3-based STREAMS driver supporting the Data Link Provider Interface dlpi(7P), on Nvidia ck8-04/mcp55/mcp51 Gigabit Ethernet controllers. The controller is a Mac chipset that works with PHY functions and provides three-speed (copper) Ethernet operation on the RJ-45 connectors.

The nge driver functions include controller initialization, frame transmit and receive, promiscuous and multicast support, and error recovery and reporting.

The nge driver and hardware support auto-negotiation, a protocol specified by the 1000 Base-T standard. Auto-negotiation allows each device to advertise its capabilities and discover those of its peer (link partner). The highest common denominator supported by both link partners is automatically selected, yielding the greatest available throughput while requiring no manual configuration. The nge driver also allows you to configure the advertised capabilities to less than the maximum (where the full speed of the interface is not required), or to force a specific mode of operation, irrespective of the link partner's advertised capabilities.

Application Programming Interface The cloning, character-special device `/dev/nge` is used to access all nge devices.

The nge driver is dependent on `/kernel/misc/mac`, a loadable kernel module that provides the DLPI and STREAMS functionality required of a LAN driver. See gld(7D) for more details on supported primitives.

You must send an explicit DL_ATTACH_REQ message to associate the opened stream with a particular device (PPA). The PPA ID is interpreted as an unsigned integer data type and indicates the corresponding device instance (unit) number. The driver returns an error (DL_ERROR_ACK) if the PPA field value does not correspond to a valid device instance number for the system. The device is initialized on first attach and de-initialized (stopped) at last detach.

The values returned by the driver in the DL_INFO_ACK primitive in response to the DL_INFO_REQ are as follows:

- Maximum SDU (with jumbo frame) is 9000. (ETHERMTU - defined in <sys/ethernet>.
- Minimum SDU is 68.
- DSLAP address length is 8 bytes.
- MAC type is DL_ETHER.
- SAP length value is -2 meaning the physical address component is followed immediately by a 2-byte sap component within the DLSAP address.
- Broadcast address value is Ethernet/IEEE broadcast address (FF:FF:FF:FF:FF:FF).

Once in the DL_ATTACHED state, you must send a DL_BIND_REQ to associate a particular Service Access Point (SAP) with the stream.

Configuration By default, the nge driver performs auto-negotiation to select the link speed and mode. Link speed and mode can be any one of the following, (as described in the *IEEE803.2* standard):

1000 Mbps, full-duplex.

1000 Mbps, half-duplex.

100 Mbps, full-duplex.

100 Mbps, half-duplex.

10 Mbps, full-duplex.

10 Mbps, half-duplex.

The auto-negotiation protocol automatically selects speed (1000 Mbps, 100 Mbps, or 10 Mbps) and operation mode (full-duplex or half-duplex) as the highest common denominator supported by both link partners. Because the nge device supports all modes, the effect is to select the highest throughput mode supported by the other device.

Alternatively, you can set the capabilities advertised by the nge device using `dladm(1M)`. The driver supports a number of parameters whose names begin with enable (see below). Each of these parameters contains a boolean value that determines whether the device advertises that mode of operation. If *en_autoneg_cap* is set to 0, the driver forces the mode of operation selected by the first non-zero parameter in priority order as listed below:

```
                (highest priority/greatest throughput)
en_1000fdx_cap         1000Mbps full duplex
en_1000hdx_cap         1000Mpbs half duplex
en_100fdx_cap          100Mpbs full duplex
en_100hdx_cap          100Mpbs half duplex
en_10fdx_cap           10Mpbs full duplex
en_10hdx_cap           10Mpbs half duplex
                      (lowest priority/least throughput)
```

For example, to prevent the device 'nge2' from advertising gigabit capabilities, enter (as super-user):

```
# dladm set-linkprop -p en_1000hdx_cap=0 nge2
# dladm set-linkprop -p en_1000fdx_cap=0 nge2
```

All capabilities default to enabled. Note that changing any capability parameter causes the link to go down while the link partners renegotiate the link speed/duplex using the newly changed capabilities.

You can obtain the current parameters settings using dladm show-linkprop. In addition, the driver exports the current state, speed, duplex setting and working mode of the link via kstat parameters (which are read only and may not be changed). For example, to check link state of device `nge0`:

```
# dladm show-linkprop -p state nge1
       LINK         PROPERTY        VALUE      DEFAULT     POSSIBLE
       nge1         state           up         up          up,down
       # dladm show-linkprop -p speed nge0
       LINK         PROPERTY        VALUE      DEFAULT     POSSIBLE
       nge1         speed           100          --        10,100,1000
       # dladm show-linkprop -p duplex nge1
       LINK         PROPERTY        VALUE      DEFAULT     POSSIBLE
       nge1         duplex          full       full        half,full
       # dladm show-linkprop -p flowctrl nge1
       LINK         PROPERTY        VALUE      DEFAULT     POSSIBLE
       nge1         flowctrl        no         bi          no,tx,rx,bi
```

The output above indicates that the link is up and running at 100Mbps full-duplex with its rx/tx direction pause capability. In addition, the driver exports its working mode by *loop_mode*. If it is set to 0, the loopback mode is disabled.

Only MCP55/CK804 chipsets accept the Maximum MTU upper to 9000 bytes. Use default_mtu to set in `/kernel/drv/nge.conf file`, then reboot to make it available. The default MTU value is 1500. For MCP55/CK804 chipsets, `nge` provides one option of minimal memory usage. Use `minimal-memory–usage = 1` in the `/kernel/drv/nge.conf` file, then reboot to make it available. With this option, the `nge` driver can reduce memory usage by two thirds. Note that setting `minimal-memory–usage = 1` does not take effect if MTU is increased above the default value. To avoid problems, do not set the `minimal-memory-usage` and `default_mtu` options together in the `nge.conf` file.

Files

`/dev/nge`	nge special character device.
`/kernel/drv/nge`	32-bit ELF Kernel module (x86).
`/kernel/drv/amd64/nge`	64-bit ELF Kernel module (x86).
`/kernel/drv/nge.conf`	Driver configuration file.

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86

See Also

`dladm(1M)`, `ndd(1M)`, `attributes(5)`, `gld(7D)`, `streamio(7I)`, `dlpi(7P)`

Writing Device Drivers

STREAMS Programming Guide

Network Interfaces Programmer's Guide

Name npe – PCI Express bus nexus driver

Description The npe nexus driver is used on X64 servers for PCI Express Root Complex devices that provide PCI Express interconnect. This driver is compliant to PCI Express base specification, Revision 1.0a.

This nexus driver provides support for the following features: Access to extended configuration space, IEEE 1275 extensions for PCI Express, Base line PCI Express error handling and PCI Express MSI interrupts.

Files

/platform/i86pc/kernel/drv/npe	32-bit ELF kernel module.
/platform/i86pc/kernel/drv/amd64/npe	64-bit ELF kernel module.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x64 PCI Express-based systems
Availability	SUNWcakr.i

See Also `attributes(5)`, `pcie(4)`, `pcie_pci(7D)`

PCI Express Base Specification v1.0a — 2003

Writing Device Drivers

IEEE 1275 PCI Bus Binding — 1998

http://playground.sun.com/1275/bindings/pci/pci-express.txt

Name ntwdt – Netra—based application watchdog timer driver

Synopsis `/dev/ntwdt`

Description The `ntwdt` driver is a multithreaded, loadable, non-STREAMS pseudo driver that provides an application with an interface for controlling a system watchdog timer.

The `ntwdt` driver implements a *virtual watchdog timer* that a privileged application (Effective UID == 0) controls via IOCTLs.

Configuration You configure the `ntwdt` driver by modifying the contents of the `ntwdt.conf` file.

Errors An `open()` fails if:

EPERM Effective user ID is not zero.

ENOENT `/dev/ntwdt` is not present or driver is not installed.

EAGAIN `/dev/ntwdt` has already been successfully open()'d.

Files

/dev/ntwdt	Special character device.
kernel/drv/sparcv9/ntwdt	SPARC ntwdt driver binary.
kernel/drv/ntwdt.conf	Driver configuraton file.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcakr.u
Architecture	SPARC

See Also `driver.conf(4)`, `attributes(5)`

Writing Device Drivers

Name ntxn – NetXen 10/1 Gigabit Ethernet network driver

Synopsis `/dev/ntxn*`

Description The `ntxn` 10/1 Gigabit Ethernet driver is a multi-threaded, loadable, clonable, GLD-based STREAMS driver supporting the Data Link Provider Interface, `dlpi(7P)`, on NetXen 10/1 Gigabit Ethernet controllers.

The `ntxn` driver functions include chip initialization, frames transmit and receive, promiscuous and multicast support, TCP and UDP checksum off-load (IPv4) and 9600 bytes jumbo frame.

The `ntxn` driver and hardware support the 10GBASE CX4, 10GBASE-SR/W, LR/W, and 10/100/1000BASE-T physical layers.

Application Programming Interface The cloning character-special device, `/dev/ntxn`, is used to access all NetXen devices installed within the system.

The `ntxn` driver is managed by the `dladm(1M)` command line utility, which allows VLANs to be defined on top of `ntxn` instances and for `ntxn` instances to be aggregated. See `dladm(1M)` for more details.

The values returned by the driver in the DL_INFO_ACK primitive in response to your DL_INFO_REQ are:

- Maximum SDU is 9600.
- Minimum SDU is 0.
- DLSAP address length is 8.
- MAC type is DL_ETHER.
- SAP (Service Access Point) length value is -2, meaning the physical address component is followed immediately by a 2-byte SAP component within the DLSAP address.
- Broadcast address value is the Ethernet/IEEE broadcast address (FF:FF:FF:FF:FF:FF).

Configuration By default, the `ntxn` driver works without any configuration file.

Files

`/dev/ntxn*`	Special character device.
`/kernel/drv/ntxn`	32–bit device driver (x86).
`/kernel/drv/amd64/ntxn`	64–bit device driver (x86).
`/kernel/drv/ntxn.conf`	Configuration file.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWntxn

Architecture	x86
Interface Stability	Committed

See Also dladm(1M), ndd(1M), netstat(1M), driver.conf(4), attributes(5), streamio(7I), dlpi(7P)

Writing Device Drivers

STREAMS Programming Guide

Network Interfaces Programmer's Guide

Name null – the null file, also called the null device

Synopsis `/dev/null`

Description Data written on the null special file, `/dev/null`, is discarded.

Reads from a null special file always return 0 bytes.

Mapping a null special file creates an address reservation of a length equal to the length of the mapping, and rounded up to the nearest page size as returned by `sysconf`(3C). No resources are consumed by the reservation. Mappings can be placed in the resulting address range via subsequent calls to `mmap` with the `-MAP_FIXED` option set.

Files `/dev/null`

See Also `mmap`(2), `sysconf`(3C)

Name nulldriver – Null driver

Description This driver succeeds probe(9E), attach(9E) and detach(9E) but provides no namespace or functionality.

In some circumstances having device nodes bound to `nulldriver` is expected. For example, prtconf(1M) might capture a nexus driver with a `nulldriver` bound child if the nexus is performing child discovery.

See Also prtconf(1M), attach(9E), detach(9E), probe(9E)

Name nv_sata – Nvidia ck804/mcp55 SATA controller driver

Synopsis

```
sata@unit-address
```

Description The `nv_sata` driver is a SATA HBA driver that supports Nvidia ck804 and mcp55 SATA HBA controllers. Note that while these Nvidia controllers support standard SATA features including SATA-II drives, NCQ, hotplug and ATAPI drives, the driver currently does not support NCQ features.

Configuration The `nv_sata` module contains no user configurable parameters.

Files

`/kernel/drv/nv_sata`
32–bit ELF kernel module (x86).

`/kernel/drv/amd64/nv_sata`
64–bit ELF kernel module (x86).

Attributes See `attributes(5)` for descriptions of the following attribute:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWnvsata

See Also `cfgadm(1M)`, `cfgadm_sata(1M)`, `prtconf(1M)`, `sata(7D)`, `sd(7D)`

Writing Device Drivers

Name nxge – Sun 10/1 Gigabit Ethernet network driver

Synopsis `/dev/nxge*`

Description The `nxge` Gigabit Ethernet driver is a multi-threaded, loadable, clonable, GLD-based STREAMS driver supporting the Data Link Provider Interface, `dlpi(7P)`, on Sun Gigabit Ethernet hardware (NIU, Sun x8, Express Dual 10 Gigabit Ethernet fiber XFP low profile adapter and the 10/100/1000BASE-T x8 Express low profile adapter).

The `nxge` driver functions include chip initialization, frame transmit and receive, flow classification, multicast and promiscuous support, and error recovery and reporting.

The `nxge` device provides fully-compliant *IEEE 802.3ae* 10Gb/s full duplex operation using XFP-based 10GigE optics (NIU, dual 10 Gigabit fiber XFP adapter). The Sun Ethernet hardware supports the *IEEE 802.3x* frame-based flow control capabilities.

For the 10/100/1000BASE-T adapter, the `nxge` driver and hardware support auto-negotiation, a protocol specified by the *1000 Base-T* standard. Auto-negotiation allows each device to advertise its capabilities and discover those of its peer (link partner). The highest common denominator supported by both link partners is automatically selected, yielding the greatest available throughput while requiring no manual configuration. The `nxge` driver also allows you to configure the advertised capabilities to less than the maximum (where the full speed of the interface is not required) or to force a specific mode of operation, irrespective of the link partner's advertised capabilities.

Application Programming Interface The cloning character-special device, `/dev/nxge`, is used to access all Sun Neptune NIU devices installed within the system.

The `nxge` driver is managed by the `dladm(1M)` command line utility, which allows VLANs to be defined on top of `nxge` instances and for `nxge` instances to be aggregated. See `dladm(1M)` for more details.

You must send an explicit DL_ATTACH_REQ message to associate the opened stream with a particular device (PPA). The PPA ID is interpreted as an unsigned integer data type and indicates the corresponding device instance (unit) number. The driver returns an error (DL_ERROR_ACK) if the PPA field value does not correspond to a valid device instance number for the system. The device is initialized on first attach and de-initialized (stopped) at last detach.

The values returned by the driver in the DL_INFO_ACK primitive in response to a DL_INFO_REQ are:

- Maximum SDU (default 1500).
- Minimum SDU (default 0). The driver pads to the mandatory 60-octet minimum packet size.
- DLSAP address length is 8.
- MAC type is `DL_ETHER`.

- SAP length value is *-2*, meaning the physical address component is followed immediately by a 2-byte SAP component within the DLSAP address.
- Broadcast address value is the Ethernet/IEEE broadcast address (FF:FF:FF:FF:FF:FF).

  Due to the nature of link address definition for IPoIB, the DL_SET_PHYS_ADDR_REQ DLPI primitive is not supported.

  In the transmit case for streams that have been put in raw mode via the DLIOCRAW ioctl, the dlpi application must prepend the 20 byte IPoIB destination address to the data it wants to transmit over-the-wire. In the receive case, applications receive the IP/ARP datagram along with the IETF defined 4 byte header.

Once in the DL_ATTACHED state, you must send a DL_BIND_REQ to associate a particular Service Access Point (SAP) with the stream.

Configuration For the *10/100/1000BASE-T* adapter, the nxge driver performs auto-negotiation to select the link speed and mode. Link speed and mode may be 10000 Mbps full-duplex (10 Gigabit adapter), 1000 Mbps full-duplex, 100 Mbps full-duplex, or 10 Mbps full-duplex, depending on the hardware adapter type. See the *IEEE802.3* standard for more information.

The auto-negotiation protocol automatically selects the 1000 Mbps, 100 Mbps, or 10 Mbps operation modes (full-duplex only) as the highest common denominator supported by both link partners. Because the nxge device supports all modes, the effect is to select the highest throughput mode supported by the other device.

You can also set the capabilities advertised by the nxge device using dladm(1M). The driver supports a number of parameters whose names begin with *en_* (see below). Each of these parameters contains a boolean value that determines if the device advertises that mode of operation. The adv_autoneg_cap parameter controls whether auto-negotiation is performed. If adv_autoneg_cap is set to 0, the driver forces the mode of operation selected by the first non-zero parameter in priority order as shown below:

```
                        (highest priority/greatest throughput)
en_1000fdx_cap            1000Mbps full duplex
en_100fdx_cap             100Mpbs full duplex
en_10fdx_cap              10Mpbs full duplex
                    (lowest priority/least throughput)
```

All capabilities default to enabled. Note that changing any capability parameter causes the link to go down while the link partners renegotiate the link speed/duplex using the newly changed capabilities.

Files

/dev/nxge*	Special character device.
/kernel/drv/nxge	32–bit device driver (x86).
/kernel/drv/sparcv9/nxge	64–bit device driver (SPARC).
/kernel/drv/amd64/nxge	64–bit device driver (x86).
/kernel/drv/nxge.conf	Configuration file.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86

See Also dladm(1M), netstat(1M), attributes(5), streamio(7I), dlpi(7P), driver.conf(4)

Writing Device Drivers

STREAMS Programming Guide

Network Interfaces Programmer's Guide

IEEE 802.3ae Specification — 2002

Name objfs – Kernel object filesystem

Description The `objfs` filesystem describes the state of all modules currently loaded by the kernel. It is mounted during boot at `/system/object`.

The contents of the filesystem are dynamic and reflect the current state of the system. Each module is represented by a directory containing a single file, 'object.' The object file is a read only ELF file which contains information about the object loaded in the kernel.

The kernel may load and unload modules dynamically as the system runs. As a result, applications may observe different directory contents in `/system/object` if they repeatedly rescan the directory. If a module is unloaded, its associated `/system/object` files disappear from the hierarchy and subsequent attempts to open them, or to read files opened before the module unloaded, elicits an error.

Files `/system/object` Mount point for objfs file system

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Private

See Also `vfstab(4)`

Notes The content of the ELF files is private to the implementation and subject to change without notice.

Name oce – Emulex OneConnect 10 GBit Ethernet Adapter Driver

Synopsis `/dev/oce*`

Description The `oce` 10 GBit ethernet adapter driver is a STREAMS based GLD (NIC driver) for 10G Ethernet functions on the Emulex OneConnect cards.

The `oce` driver initializes the NIC functions on the chip and implements send/receive of frames. The driver provides statistics and error reporting. The driver also supports multicast and promiscuous modes for send/receive, VLANs, lso, and so forth. The driver supports `mtu` of 1500 or 9000.

Configuration The device can be configured using tools such as `dladm` or `ifconfig`.

The `mtu` can be changed using the `dladm set-linkprop` command:

```
dladm set-linkprop -p mtu=9000 oce0
```

The only valid value for speed/mode is 10 Gbps/full-duplex.

The interfaces created by the `oce` driver can be configured through `ifconfig`:

```
ifconfig oce0 plumb xxx.xxx.xxx.xxx up ifconfig oce0 down unplumb
```

Files

`/kernel/drv/oce`	32-bit ELF kernel module
`/kernel/drv/amd64/oce`	64-bit ELF kernel module, x86
`/kernel/drv/sparcv9/oce`	64-bit ELF kernel module, SPARC

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SPARC, x86
Availability	SUNWemlxs

See Also `dladm(1M)`, `ifconfig(1M)`, `netstat(1M)`, `prtconf(1M)`, `attributes(5)`, `dlpi(7P)`

Writing Device Drivers

Network Interface Guide

STREAMS Programming Guide

IEEE 802.3ae Specificiation, IEEE - 2002

Name ohci – OpenHCI host controller driver

Synopsis `usb@unit-address`

Description The `ohci` driver is a USBA (Solaris USB Architecture) compliant nexus driver that supports the *Open Host Controller Interface Specification 1.1*, an industry standard developed by Compaq, Microsoft, and National Semiconductor.

The `ohci` driver supports bulk, interrupt, control and isochronous transfers.

Files

`/kernel/drv/ohci`	32–bit x86 ELF kernel module
`/kernel/drv/amd64/ohci`	64–bit x86 ELF kernel module
`/kernel/drv/sparcv9/ohci`	64–bit SPARC ELF kernel module
`/kernel/drv/ohci.conf`	driver configuration file

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86, PCI-based systems
Availability	SUNWusb

See Also `attributes(5)`, `ehci(7D)`, `hubd(7D)`, `uhci(7D)`, `usba(7D)`

Writing Device Drivers

Universal Serial Bus Specification 2.0

Open Host Controller Interface Specification for USB 1.0a

System Administration Guide: Basic Administration

http://www.sun.com/io

Diagnostics All host controller errors are passed to the client drivers. Root hub errors are documented in `hubd(7D)`.

In addition to being logged, the following messages may appear on the system console. All messages are formatted in the following manner:

```
WARNING: <device path> <ohci><instance number>>: Error message...
```

Unrecoverable USB Hardware Error.
: There was an unrecoverable USB hardware error reported by the OHCI Controller. Please reboot the system. If this problem persists, contact your system vendor.

No SOF interrupts have been received. This OHCI USB controller is unusable.
: The USB hardware is not generating Start Of Frame interrupts. Please reboot the system. If this problem persists, contact your system vendor.

The following messages may be entered into the system log. They are formatted in the following manner:

```
<device path> <ohci<instance number>): Message...
```

Failed to program frame interval register.	For an unspecified reason, the frame interval register has been nulled out by the Uli M1575 chip. Please reboot the system. If this problem persists, contact your system vendor.

Name openprom – PROM monitor configuration interface

Synopsis

```
#include <sys/fcntl.h>

#include <sys/types.h>

#include <sys/openpromio.h>

open("/dev/openprom", mode);
```

Description The internal encoding of the configuration information stored in EEPROM or NVRAM varies from model to model, and on some systems the encoding is "hidden" by the firmware. The `openprom` driver provides a consistent interface that allows a user or program to inspect and modify that configuration, using `ioctl(2)` requests. These requests are defined in `<sys/openpromio.h>`:

```
struct openpromio {
    uint_t  oprom_size;       /* real size of following data */
    union {
         char  b[1];          /* NB: Adjacent, Null terminated */
         int   i;
               } opio_u;
};
#define oprom_array opio_u.b  /* property name/value array */
#define oprom_node opio_u.i   /* nodeid from navigation config-ops */
#define oprom_len opio_u.i    /* property len from OPROMGETPROPLEN */
#define OPROMMAXPARAM 32768   /* max size of array (advisory) */
```

For all `ioctl(2)` requests, the third parameter is a pointer to a `struct openpromio`. All property names and values are null-terminated strings; the value of a numeric option is its ASCII representation.

For the raw `ioctl(2)` operations shown below that explicitly or implicitly specify a nodeid, an error may be returned. This is due to the removal of the node from the firmware device tree by a Dynamic Reconfiguration operation. Programs should decide if the appropriate response is to restart the scanning operation from the beginning or terminate, informing the user that the tree has changed.

ioctls

`OPROMGETOPT` This ioctl takes the null-terminated name of a property in the *oprom_array* and returns its null-terminated value (overlaying its name). *oprom_size* should be set to the size of *oprom_array*; on return it will contain the size of the returned value. If the named property does not exist, or if there is not enough space to hold its value, then *oprom_size* will be set to zero. See `BUGS` below.

`OPROMSETOPT` This ioctl takes two adjacent strings in *oprom_array*; the null-terminated property name followed by the null-terminated value.

`OPROMSETOPT2` This ioctl is similar to `OPROMSETOPT`, except that it uses the difference between the actual user array size and the length of the property name plus its null terminator.

`OPROMNXTOPT` This ioctl is used to retrieve properties sequentially. The null-terminated name of a property is placed into *oprom_array* and on return it is replaced with the null-terminated name of the next property in the sequence, with *oprom_size* set to its length. A null string on input means return the name of the first property; an *oprom_size* of zero on output means there are no more properties.

`OPROMNXT`
`OPROMCHILD`
`OPROMGETPROP`
`OPROMNXTPROP` These ioctls provide an interface to the raw *config_ops* operations in the PROM monitor. One can use them to traverse the system device tree; see `prtconf(1M)`.

`OPROMGETPROPLEN` This ioctl provides an interface to the *property length* raw config op. It takes the name of a property in the buffer, and returns an integer in the buffer. It returns the integer `-1` if the property does not exist; `0` if the property exists, but has no value (a boolean property); or a positive integer which is the length of the property as reported by the PROM monitor. See BUGS below.

`OPROMGETVERSION` This ioctl returns an arbitrary and platform-dependent NULL-terminated string in *oprom_array*, representing the underlying version of the firmware.

Errors

`EAGAIN` There are too many opens of the `/dev/openprom` device.

`EFAULT` A bad address has been passed to an `ioctl(2)` routine.

`EINVAL` The size value was invalid, or (for `OPROMSETOPT`) the property does not exist, or an invalid ioctl is being issued, or the ioctl is not supported by the firmware, or the nodeid specified does not exist in the firmware device tree.

`ENOMEM` The kernel could not allocate space to copy the user's structure.

`EPERM` Attempts have been made to write to a read-only entity, or read from a write only entity.

`ENXIO` Attempting to open a non-existent device.

Examples

EXAMPLE 1 *oprom_array* Data Allocation and Reuse

The following example shows how the *oprom_array* is allocated and reused for data returned by the driver.

EXAMPLE 1 *oprom_array* Data Allocation and Reuse *(Continued)*

```
/*
 * This program opens the openprom device and prints the platform
 * name (root node name property) and the prom version.
 *
 * NOTE: /dev/openprom is readable only by user 'root' or group 'sys'.
 */
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/openpromio.h>
#define    min(a, b)    (a < b ? a : b)
#define    max(a, b)    (a > b ? a : b)
#define MAXNAMESZ 32          /* Maximum property *name* size */
#define BUFSZ 1024            /* A Handly default buffer size */
#define MAXVALSZ    (BUFSZ - sizeof (int))
static char *promdev = "/dev/openprom";
/*
 * Allocate an openpromio structure big enough to contain
 * a bufsize'd oprom_array. Zero out the structure and
 * set the oprom_size field to bufsize.
 */
static struct openpromio *
opp_zalloc(size_t bufsize)
{
    struct openpromio *opp;
    opp = malloc(sizeof (struct openpromio) + bufsize);
    (void) memset(opp, 0, sizeof (struct openpromio) + bufsize);
    opp->oprom_size = bufsize;
    return (opp);
}
/*
 * Free a 'struct openpromio' allocated by opp_zalloc
 */
static void
opp_free(struct openpromio *opp)
{
    free(opp);
}
/*
 * Get the peer node of the given node.  The root node is the peer of zero.
 * After changing nodes, property lookups apply to that node.  The driver
 * 'remembers' what node you are in.
```

EXAMPLE 1 *oprom_array* Data Allocation and Reuse *(Continued)*

```
 */
static int
peer(int nodeid, int fd)
{
    struct openpromio *opp;
    int i;
    opp = opp_zalloc(sizeof (int));
    opp->oprom_node = nodeid;
    if (ioctl(fd, OPROMNEXT, opp) < 0) {
        perror("OPROMNEXT");
        exit(1);
    }
    i = opp->oprom_node;
    opp_free(opp);
    return(i);
}
int
main(void)
{
    struct openpromio *opp;
    int fd, proplen;
    size_t buflen;
    if ((fd = open(promdev, O_RDONLY)) < 0)  {
        fprintf(stderr, "Cannot open openprom device\n");
        exit(1);
    }
    /*
     * Get and print the length and value of the
     * root node 'name' property
     */
    (void) peer(0, fd);        /* Navigate to the root node */
    /*
     * Allocate an openpromio structure sized big enough to
     * take the string "name" as input and return the int-sized
     * length of the 'name' property.
     * Then, get the length of the 'name' property.
     */
    buflen = max(sizeof (int), strlen("name") + 1);
    opp = opp_zalloc(buflen);
    (void) strcpy(opp->oprom_array, "name");
    if (ioctl(fd, OPROMGETPROPLEN, opp) < 0) {
        perror("OPROMGETPROPLEN");
        /* exit(1); */
        proplen = 0;    /* down-rev driver? */
    } else
```

EXAMPLE 1 *oprom_array* Data Allocation and Reuse *(Continued)*

```
        proplen = opp->oprom_len;
    opp_free(opp);
    if (proplen == -1) {
        printf("'name' property does not exist!\n");
        exit (1);
    }
    /*
     * Allocate an openpromio structure sized big enough
     * to take the string 'name' as input and to return
     * 'proplen + 1' bytes.  Then, get the value of the
     * 'name' property. Note how we make sure to size the
     * array at least one byte more than the returned length
     * to guarantee NULL termination.
     */
    buflen = (proplen ? proplen + 1 : MAXVALSZ);
    buflen = max(buflen, strlen("name") + 1);
    opp = opp_zalloc(buflen);
    (void) strcpy(opp->oprom_array, "name");
    if (ioctl(fd, OPROMGETPROP, opp) < 0) {
        perror("OPROMGETPROP");
        exit(1);
    }
    if (opp->oprom_size != 0)
        printf("Platform name <%s> property len <%d>\n",
            opp->oprom_array, proplen);
    opp_free(opp);
    /*
     * Allocate an openpromio structure assumed to be
     * big enough to get the 'prom version string'.
     * Get and print the prom version.
     */
    opp_zalloc(MAXVALSZ);
    opp->oprom_size = MAXVALSZ;
    if (ioctl(fd, OPROMGETVERSION, opp) < 0) {
        perror("OPROMGETVERSION");
        exit(1);
    }
    printf("Prom version <%s>\n", opp->oprom_array);
    opp_free(opp);
    (void) close(fd);
    return (0);
}
```

Files `/dev/openprom` PROM monitor configuration interface

See Also eeprom(1M), monitor(1M), prtconf(1M), ioctl(2), mem(7D)

Bugs There should be separate return values for non-existent properties as opposed to not enough space for the value.

An attempt to set a property to an illegal value results in the PROM setting it to some legal value, with no error being returned. An `OPROMGETOPT` should be performed after an `OPROMSETOPT` to verify that the set worked.

Some PROMS *lie* about the property length of some string properties, omitting the NULL terminator from the property length. The `openprom` driver attempts to *transparently* compensate for these bugs when returning property values by NULL terminating an extra character in the user buffer if space is available in the user buffer. This extra character is excluded from the *oprom_size* field returned from `OPROMGETPROP` and `OPROMGETOPT` and excluded in the *oprom_len* field returned from `OPROMGETPROPLEN` but is returned in the user buffer from the calls that return data, if the user buffer is allocated at least one byte larger than the property length.

Name oplkmdrv – key management driver for the SPARC Enterprise Server family

Synopsis `kmdrv`

Description `oplkmdrv` is a character driver that implements a framework for exchanging the security keys with the Service Processor on a SPARC Enterprise Server. The `oplkmdrv` driver is specific to the SPARC Enterprise Server family.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWdscpr
Interface Stability	Private

`attributes(5)`

See Also `attributes(5)`

Name oplmsu – Serial I/O multiplexing STREAMS device driver

Synopsis `/pseudo-console`

Description The `oplmsu` driver is a STREAMS multiplexer driver that connects multiple serial devices to the system console.

Currently, this support is provided only on a SPARC Enterprise Server.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcakr
Interface Stability	Private

See Also `attributes(5)`

Name oplpanel – device driver for the SPARC Enterprise Server family

Description The `oplpanel` device driver monitors the panel reset button. If the button is pressed, a high-level interrupt is generated, and the `oplpanel` driver causes a system panic.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcakr
Interface Stability	Private

See Also `attributes(5)`

Name packet, PF_PACKET – packet interface on device level

Synopsis

```
#include <sys/socket.h>

#include <netpacket/packet.h>

#include <sys/ethernet.h>

packet_socket = socket(2,7,n)(PF_PACKET, int socket_type, int protocol);
```

Description Packet sockets are used to receive or send (2,n) raw (3x,7, 8,3x cbreak) packets at the device driver (OSI Layer 2) level. These allow users to implement protocol modules in (1,8) user space on top of the physical layer.

The `socket_type` is either `SOCK_RAW` for raw (3x,7,8,3x cbreak) packets including the link (1,2) level header or `SOCK_DGRAM` for cooked packets with the link (1,2) level header removed. The link (1,2) level header information is available in (1,8) a common format in (1,8) a `sockaddr_ll`. protocol is the IEEE 802.3 protocol number in (1,8) network order. See the `<sys/ethernet.h>` include file (1,n) for a list of allowed protocols. When protocol is set (7,n,1 builtins) to `htons` (`ETH_P_ALL`) then all protocols are received. All incoming packets of that protocol type is passed to the packet socket (2,7,n) before they are passed to the protocols implemented in (1,8) the kernel.

Only process with the `net_rawaccesss` privilege may create `PF_PACKET` sockets. Processes in the global zone may bind to any network interface that is displayed using the command: `dladm show-link`.

`SOCK_RAW` packets are passed to and from the device driver without any changes in (1,8) the packet data. When receiving a packet, the address is still parsed and passed in (1,8) a standard `sockaddr_ll` address structure. When transmitting a packet, the user supplied buffer should contain the physical layer header. That packet is then queued unmodified to the network driver of the interface defined by the destination address.

`SOCK_DGRAM` operates on a slightly higher level. The physical header is removed before the packet is passed to the user. Packets sent through a `SOCK_DGRAM` packet socket (2,7,n) get a suitable physical layer header based on the information in (1,8) the `sockaddr_ll` destination address before they are queued.

By default, all packets of the specified protocol type are passed to a packet socket. To only get packets from a specific interface use bind (2,n,1 builtins)(2) specifying an address in (1,8) a `struct sockaddr_ll` to bind (2,n,1 builtins) the packet socket (2,7,n) to an interface. Only the `sll_protocol` and the `sll_ifindex` address fields are used for purposes of binding.

The `connect(3SOCKET)` operation is not supported on packet sockets.

Address Types The `sockaddr_ll` is a device independent physical layer address.

```
struct sockaddr_ll {
   unsigned short sll_family; /* Always AF_PACKET */
   unsigned short  sll_protocol;  /* Physical layer protocol */
```

```
   int             sll_ifindex;   /* Interface number */
   unsigned short  sll_hatype;    /* Header type */
   unsigned char   sll_pkttype;   /* Packet type */
   unsigned char   sll_halen;     /* Length of address */
   unsigned char   sll_addr[8];   /* Physical layer address */
};
```

`sll_protocol` is the standard ethernet protocol type in (1,8) network order as defined in (1,8) the `sys/ethernet.h` include file. It defaults to the socket (2,7,n)'s protocol. `sll_ifindex` is the interface index of the interface. See netdevice(7). 0 matches any interface (only legal for binding). `sll_hatype` is a ARP type as defined in (1,8) the `sys/ethernet.h` include file. `sll_pkttype` contains the packet type. Valid types are `PACKET_HOST` for a packet addressed to the local host(1,5), `PACKET_BROADCAST` for a physical layer broadcast packet, `PACKET_MULTICAST` for a packet sent to a physical layer multicast address, `PACKET_OTHERHOST` for a packet to some other host (1,5) that has been caught by a device driver in(1,8) promiscuous mode, and `PACKET_OUTGOING` for a packet originated from the local host(1,5) that is looped back to a packet socket. These types make only sense for receiving. `sll_addr` and `sll_halen` contain the physical layer, for example, IEEE 802.3, address and its length. The exact interpretation depends on the device.

When you send (2,n) packets it is enough to specify `sll_family`, `sll_addr`, `sll_halen`, `sll_ifindex`. The other fields should be 0. `sll_hatype` and `sll_pkttype` are set (7,n,1 builtins) on received packets for your information. For bind (2,n,1 builtins) only `sll_protocol` and `sll_ifindex` are used.

Socket Options

Packet sockets can be used to configure physical layer multicasting and promiscuous mode. It works by calling `setsockopt(3SOCKET)` on a packet socket (2,7,n) for `SOL_PACKET` and one of the options `PACKET_ADD_MEMBERSHIP` to add a binding or `PACKET_DROP_MEMBERSHIP` to drop it. They both expect a `packet_mreq` structure as argument:

```
struct packet_mreq
{
   int             mr_ifindex;    /* interface index */
   unsigned short  mr_type;       /* action */
   unsigned short  mr_alen;       /* address length */
   unsigned char   mr_address[8]; /* physical layer address */
};
```

`mr_ifindex` contains the interface index for the interface whose status should be changed. The `mr_type` parameter specifies which action to perform. `PACKET_MR_PROMISC` enables receiving all packets on a shared medium often known as *promiscuous mode*, `PACKET_MR_MULTICAST` binds the socket (2,7,n) to the physical layer multicast group specified in (1,8) `mr_address` and `mr_alen`. `PACKET_MR_ALLMULTI` sets the socket (2,7,n) up to receive all multicast packets arriving at the interface.

In addition the traditional ioctls, `SIOCSIFFLAGS`, `SIOCADDMULTI`, and `SIOCDELMULTI` can be used for the same purpose.

See Also connect(3SOCKET), setsockopt(3SOCKET)

Notes For portable programs it is suggested to usepcap(3C) instead of PF_PACKET; although this only covers a subset of the PF_PACKET features.

The SOCK_DGRAM packet sockets make no attempt to create or parse the IEEE 802.2 LLC header for a IEEE 802.3 frame. When ETH_P_802_3 is specified as protocol for sending the kernel creates the 802.3 frame and fills out the length field; the user has to supply the LLC header to get a fully conforming packet. Incoming 802.3 packets are not multiplexed on the DSAP/SSAP protocol fields; instead they are supplied to the user as protocol ETH_P_802_2 with the LLC header prepended. It is therefore not possible to bind (2,n,1 builtins) to ETH_P_802_3; bind (2,n,1 builtins) to ETH_P_802_2 instead and do the protocol multiplex yourself. The default for sending is the standard Ethernet DIX encapsulation with the protocol filled in.

Packet sockets are not subject to the input or output firewall chains.

Name pcan – Cisco Aironet 802.11b wireless NIC driver

Description The `pcan` wireless NIC driver is a multi-threaded, loadable, clonable, GLDv3-based STREAMS driver. It supports the pccard and PCI/MiniPCI cards with the Cisco Aironet *802.11b* chipset. For pccard, the driver works in both SPARC and x86 (32–bit/64–bit) modes. For PCI/MiniPCI card, the driver works in 32–bit x86 mode only.

Driver Configuration The `pcan` driver supports *802.11b* data rates of 1, 2, 5.5 and 11 (Mbits/sec). The default is 11.

The `pcan` driver supports BSS networks (also known as "ap" or "infrastructure" networks) and IBSS networks (also known as "ad-hoc" networks). For authentication type, the `pcan` driver supports the "open" (or "open-system") mode. For encryption type, only WEP is currently supported. You perform configuration and administration tasks using the `dladm(1M)` and `wificonfig(1M)` utilities.

Files

`/dev/pcan*`	Special character device.
/kernel/drv/pcan	32-bit ELF kernel module (x86).
`/kernel/drv/amd64/pcan`	64-bit ELF kernel module (x86).
`/kernel/drv/sparcv9/pcan`	64-bit ELF kernel module (SPARC).

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SPARC, x86

See Also `dladm(1M)`, `wificonfig(1M)`, `attributes(5)`, `gld(7D)`

802.11b Standard for Wireless Local Area Networks (WLANs) - IEEE

Name pcata – PCMCIA ATA card device driver

Synopsis

```
pcata@socket#:a -u

pcata@socket#:a -u,raw
```

Description The PCMCIA ATA card device driver supports PCMCIA ATA disk and flash cards that follow the following standards:

- PC card 2.01 compliance (`MBR+fdisk` table required for all platforms).
- PC card ATA 2.01 compliance.
- PC card services 2.1 compliance.

The driver supports standard PCMCIA ATA cards that contain a Card Information Structure (CIS). For PCMCIA, nodes are created in `/devices` that include the socket number as one component of the device name referred to by the node. However, the names in `/dev`, `/dev/dsk`, and `/dev/rdsk` follow the current conventions for ATA devices, which do not encode the socket number in any part of the name. For example, you may have the following:

Platform	/devices name	/dev/dsk name
x86	/devices/isa/pcic@1,3e0 /disk@0:a	/dev/dsk/c1d0s0
SPARC	/devices/iommu@f,e0000000 /sbus@f,e0001000 /SUNW, pcmcia@3,0 /disk@0:a	/dev/dsk/c1d0s0

Preconfiguration If a PC Card ATA device is recognized, the pcata driver is automatically loaded, IRQs allocated, devices nodes created, and special files created (if they do not already exist).

Known Problems and Limitations

- You need to umount the file system before removing the disk.
- The `ufs` file systems on removable media (PC Card ATA) should have one of the `onerror={panic, lock, umount}` mount options set.

Configuration Configuration topics include initial installation and configuration, identifying an unrecognized device, special files and hot-plugging.

Initial Installation and Configuration

1. Install the Solaris software.
2. Boot the system.
3. Insert the PC card ATA device.

Identifying an Unrecognized Device

If you insert a PC card ATA device and it is not recognized (no special files created), use the `prtconf` command to identify the problem.

1. Run the `prtconf -D` command to see if your `pcata` card is recognized. (A recognized device will appear at the end of the `prtconf` output. For example:

```
# prtconf -D
. . .
```

```
pcic, instance #0 (driver name: pcic)
    .  .  .
   disk, instance #0
```

2. If `pcata` does not appear in the `prtconf` output, there is a problem with the PC card adapter configuration or with the hardware. Check to see whether the problem is with the card or the adapter by trying to use the card on another machine and by seeing if it works on the same machine using DOS.

Special Files For PC card devices, nodes are created in `/devices` that include the socket number as one component of a device name that the node refers to. However, the `/prtc/dev` names and the names in `/dev/dsk` and `/dev/rdsk` do follow the current convention for ATA devices, which do not encode the socket number in any part of the name.

Hot-Plugging

- If you want to remove the disk, you must unmount the file system.
- Use the `mkfs_pcfs(1M)` command to create a `pcfs` file system:

  ```
  # mkfs -F pcfs /dev/rdsk/c#d#p0:d
  ```

- To mount a `pcfs` file system, type:

  ```
  # mount -F pcfs /dev/dsk/c#d#p0:c /mnt
  ```

- If you want to create a `ufs` file system, use the `newfs` command and type:

  ```
  # newfs /dev/rdsk/c#d#s#
  ```

- To mount a `ufs` file system, type:

  ```
  # mount -F ufs /dev/dsk/c#d#s# /mnt
  ```

- To create a Solaris partition, run the format command and go to the Partition menu. For more information, see the `format(1M)` man page.

Files `/kernel/drv/pcata` `pcata` driver

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWpsdpr

See Also `format(1M)`, `mount(1M)`, `newfs(1M)`, `pcmcia(7D)`, `attributes(5)`, `pcfs(7FS)`

Name pcfs – FAT formatted file system

Synopsis

```
#include <sys/param.h>
#include <sys/mount.h>
#include <sys/fs/pc_fs.h>

int mount(const char *spec,
     const char * dir, int  mflag,
     "pcfs", NULL, 0, char *optptr,
     int optlen);
```

Description `pcfs` is a file system type that enables direct access to files on FAT formatted disks from within the SunOS operating system.

Once mounted, `pcfs` provides standard SunOS file operations and semantics. Using `pcfs`, you can create, delete, read, and write files on a FAT formatted disk. You can also create and delete directories and list files in a directory.

`pcfs` supports FAT12 (floppies) and FAT16 and FAT32 file systems.

`pcfs` file systems can be force umounted using the `-f` argument to `umount(1M)`.

The `pcfs` file system contained on the block special file identified by *spec* is mounted on the directory identified by *dir*. *spec* and *dir* are pointers to pathnames. *mflag* specifies the `mount` options. The MS_DATA bit in *mflag* must be set. Mount options can be passed to `pcfs` using the optptr and optlen arguments. See `mount_pcfs(1M)` for a list of mount options supported by `pcfs`.

Because FAT formatted media can record file timestamps between January 1st 1980 and December 31st 2127, it's not possible to fully represent UNIX *time_t* in `pcfs` for 32 bit or 64 bit programs. In particular, if post-2038 timestamps are present on a FAT formatted medium and `pcfs` returns these, 32bit applications may unexpectedly fail with `EOVERFLOW` errors. To prevent this, the default behaviour of `pcfs` has been modified to clamp post-2038 timestamps to the latest possible value for a 32bit *time_t*, which is January 19th 2038, 03:14:06 UTC when setting and retrieving file timestamps. You can override this behavior using the `noclamptime` mount option, as described in `mount_pcfs(1M)`.

Timestamps on FAT formatted media are recorded in local time. If the recording and receiving systems use different timezones, the representation of timestamps shown on the two systems for the same medium might vary. To correct this, `pcfs` provides a timezone mount option to force interpretation of timestamps as read from a FAT formatted medium in a given timezone (that of the recorder). By default, the local timezone of the receiver is used. See `mount_pcfs(1M)` for details.

The root directory of a FAT formatted medium has no timestamps and `pcfs` returns the time when the mount was done as timestamp for the root of the filesystem.

The FAT filesystem doesn't support multiple links. As a result, the link count for all files and directories in `pcfs` is hard-coded as `1`.

Mounting File Systems

Use the following command to mount `pcfs` from diskette:

`mount -F pcfs` *device-special directory-name*

You can use:

`mount` *directory-name*

if the following line is in your `/etc/vfstab` file:

device-special `-` *directory-name*`pcfs – no rw`

Use the following command to mount `pcfs` from non-diskette media:

`mount -F pcfs` *device-special*`:`*logical-drive* `directory-name`

You can use:

`mount` *directory-name*

if the following line is in your `/etc/vfstab` file:

device-special`:`*logical_drive* `–` *directory-name* `pcfs – no rw`

device-special specifies the special block device file for the diskette (`/dev/diskette`*N*) or the entire hard disk (`/dev/dsk/c`*N*`t`*N*`d`*N*`p0` for a SCSI disk, and `/dev/dsk/c`*N*`d`*N*`p0` for IDE disks) or the PCMCIA pseudo-floppy memory card (`/dev/dsk/c`*N*`t`*N*`d`*N*`s`*N*).

logical-drive specifies either the DOS logical drive letter (`c` through `z`) or a drive number (`1` through `24`). Drive letter `c` is equivalent to drive number `1` and represents the Primary DOS partition on the disk; drive letters `d` through `z` are equivalent to drive numbers `2` through `24`, and represent DOS drives within the Extended FAT partition. Note that *device-special* and *logical-drive* must be separated by a colon.

directory-name specifies the location where the file system is mounted.

For example, to mount the Primary DOS partition from a SCSI hard disk, use:

```
mount -F pcfs /dev/dsk/cNtNdNp0:c /pcfs/c
```

To mount the first logical drive in the Extended DOS partition from an IDE hard disk, use:

```
mount -F pcfs /dev/dsk/cNdNp0:d /pcfs/d
```

To mount a DOS diskette in the first floppy drive when volume management is not running use:

```
mount -F pcfs /dev/diskette /pcfs/a
```

If Volume Management is running, run `volcheck(1)` to automatically mount the floppy and some removable disks.

To mount a PCMCIA pseudo-floppy memory card, with Volume Management not running (or not managing the PCMCIA media), use:

```
mount -F pcfs /dev/dsk/cNtNdNsN /pcfs
```

Conventions

Files and directories created through `pcfs` must comply with either the FAT short file name convention or the long file name convention introduced with Windows 95. The FAT short file name convention is of the form *filename*[.*ext*], where *filename* generally consists of from one to eight upper-case characters, while the optional *ext* consists of from one to three upper-case characters.

The long file name convention is much closer to Solaris file names. A long file name can consist of any characters valid in a short file name, lowercase letters, non-leading spaces, the characters `+,;=[]`, any number of periods, and can be up to 255 characters long. Long file names have an associated short file name for systems that do not support long file names (including earlier releases of Solaris). The short file name is not visible if the system recognizes long file names. `pcfs` generates a unique short name automatically when creating a long file name.

Given a long file name such as `This is a really long filename.TXT`, the short file name will generally be of the form `THISIS~`*N*`.TXT`, where *N* is a number. The long file name will probably get the short name `THISIS~1.TXT`, or `THISIS~2.TXT` if `THISIS~1.TXT` already exits (or `THISIS~3.TXT` if both exist, and so forth). If you use `pcfs` file systems on systems that do not support long file names, you may want to continue following the short file name conventions. See `EXAMPLES`.

When creating a file name, `pcfs` creates a short file name if it fits the FAT short file name format, otherwise it creates a long file name. This is because long file names take more directory space. Because the root directory of a `pcfs` file system is fixed size, long file names in the root directory should be avoided if possible.

When displaying file names, `pcfs` shows them exactly as they are on the media. This means that short names are displayed as uppercase and long file names retain their case. Earlier versions of `pcfs` folded all names to lowercase, which can be forced with the `PCFS_MNT_FOLDCASE` mount option. All file name searches within `pcfs`, however, are treated as if they were uppercase, so `readme.txt` and `ReAdMe.TxT` refer to the same file.

To format a diskette or a PCMCIA pseudo-floppy memory card in FAT format in the SunOS system, use either the `fdformat -d` or the DOS `FORMAT` command.

Boot Partitions

On x86 systems, hard drives may contain an `fdisk` partition reserved for the Solaris boot utilities. These partitions are special instances of `pcfs`. You can mount an x86 boot partition with the command:

```
mount -F pcfs device-special:boot directory-name
```

or you can use:

```
mount directory-name
```

if the following line is in your `/etc/vfstab` file:

device-special`:boot` — *directory-name* `pcfs — no rw`

device-special specifies the special block device file for the entire hard disk (`/dev/dsk/c`*N*`t`*N*`d`*N*`p0`)

directory-name specifies the location where the file system is mounted.

All files on a boot partition are owned by super-user. Only the super-user may create, delete, or modify files on a boot partition.

Examples

EXAMPLE 1 Sample Displays of File Names

If you copy a file `financial.data` from a UNIX file system to `pcfs`, it displays as `financial.data` in `pcfs`, but may show up as `FINANC~1.DAT` in systems that do not support long file names.

The following are legal long file names. They are also *illegal* short file names:

```
test.sh.orig
data+
.login
```

Other systems that do not support long file names may see:

```
TESTSH~1.ORI
DATA~1
LOGIN~1
```

The short file name is generated from the initial characters of the long file name, so differentiate names in the first few characters. For example, these names:

```
WorkReport.January.Data
WorkReport.February.Data
WorkReport.March.Data
```

result in these short names, which are not distinguishable:

```
WORKRE~1.DAT
WORKRE~2.DAT
WORKRE~13.DAT
```

These names, however:

```
January.WorkReport.Data
```

EXAMPLE 1 Sample Displays of File Names *(Continued)*

```
February.WorkReport.Data
March.WorkReport.Data
```

result in the more descriptive short names:

```
JANUAR~1.DAT
FEBRUA~1.DAT
MARCHW~1.DAT
```

Files

`/usr/lib/fs/pcfs/mount`	`pcfs mount` command
`/usr/kernel/fs/pcfs`	32-bit kernel module

Environment Variables See `environ(5)` for descriptions of the following environment variables for the current locale setting: LANG, LC_ALL, LC_CTYPE, and LC_COLLATE.

See Also `chgrp(1)`, `chown(1)`, `dos2unix(1)`, `eject(1)`, `fdformat(1)`, `unix2dos(1)`, `volcheck(1)`, `mount(1M)`, `mount_pcfs(1M)`, `umount(1M)`, `ctime(3C)`, `vfstab(4)`, `environ(5)`,

Warnings Do not physically eject a FAT floppy while the device is mounted as `pcfs`. If Volume Management is managing a device, use the `eject(1)` command before physically removing media.

When mounting `pcfs` on a hard disk, make sure the first block on that device contains a valid `fdisk` partition table.

Because `pcfs` has no provision for handling owner-IDs or group-IDs on files, `chown(1)` or `chgrp(1)` may generate various errors. This is a limitation of `pcfs`, but it should not cause problems other than error messages.

Notes Only the following characters are allowed in `pcfs` short file names and extensions:

```
0-9
A-Z
$#&@!%()-{}<>‘_^~|’
```

SunOS and FAT use different character sets and have different requirements for the text file format. Use the `dos2unix(1)` and `unix2dos(1)` commands to convert files between them.

`pcfs` offers a convenient transportation vehicle for files between Sun workstations and PCs. Because the FAT disk format was designed for use under DOS, it does not operate efficiently under the SunOS system and should not be used as the format for a regular local storage. Instead, use `ufs` for local storage within the SunOS system.

Although long file names can contain spaces (just as in UNIX file names), some utilities may be confused by them.

This implementation of `pcfs` conforms to the behavior exhibited by Windows 95 version 4.00.950.

When `pcfs` encounters long file names with non-ASCII characters, it converts such long file names in Unicode scalar values into UTF-8 encoded filenames so that they are legible and usable with any of Solaris UTF-8 locales. In the same context, when new file names with non-ASCII characters are created, `pcfs` expects that such file names are in UTF-8. This feature increases the interoperability of `pcfs` on Solaris with other operating systems.

Bugs `pcfs` should handle the disk change condition in the same way that DOS does, so you do not need to unmount the file system to change floppies.

Name pcic – Intel i82365SL PC Card Interface Controller

Description The Intel i82365SL PC Card interface controller provides one or more PCMCIA PC card sockets. The `pcic` driver implements a PCMCIA bus nexus driver.

The driver provides basic support for the Intel 82365SL and compatible chips. Tested chips are:

- Intel — 82365SL.
- Cirrus Logic — PD6710/PD6720/PD6722.
- Vadem — VG365/VG465/VG468/VG469.
- Toshiba — PCIC and ToPIC
- Ricoh — RF5C366/RL5C466/RL5C475/RL5C476/RL5C477/RL5C478.
- 02Micro — OZ6912/6972.
- Texas Instruments — PCI1130/PCI1131/PCI1031/PCI1221/PCI1225/PCI1520/PCI 1410/PCI1420/PCI4520/PCI7510/PCI7621.

While most systems using one of the above chips will work, some systems are not supported due to hardware designs options that may not be software detectable.

Direct access to the PCMCIA hardware is not supported. All device access must be through the DDI.

Configuration Configuration of PC Card interface controllers are automatically done in the system by leveraging ACPI on x86 (or OBP on SPARC). Configuration includes allocation of device memory, I/O ports, CardBus subordinary bus number and interrupts. There is no user-interference required. Note that the controller may not work when ACPI is disabled.

There is one driver configuration property defined in the `pcic.conf` file:

`interrupt-priorities=6;` This property must be defined and must be below 10.

Files

`/kernel/drv/pcic`	`pcic` driver
`/kernel/drv/pcic.conf`	`pcic` configuration file

See Also `cardbus(4)`, `pcmcia(7D)`

Name pcicmu – PCI bus nexus driver for the SPARC Enterprise Server family

Description The `pcicmu` nexus driver is used for onboard devices for the SPARC Enterprise Server family.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcakr
Interface Stability	Private

See Also `attributes(5)`

Name pcie_pci – PCI Express bridge nexus driver

Description The `pcie_pci` nexus driver is used on X64 servers for PCI Express bridge class devices including PCI Express root ports which are implemented as virtual bridges and PCI Express to PCI/PCI-X bridges.

The `pcie_pci` driver is compliant with the *PCI Express Base*, Revision 1.0a specification and supports Base line PCI Express error handling and PCI Express Hot Plug.

Files

/platform/i86pc/kernel/drv/pcie_pci	32-bit ELF kernel module.
/platform/i86pc/kernel/drv/amd64/pcie_pci	64-bit ELF kernel module.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x64 PCI Express-based systems
Availability	SUNWcakr.i

See Also `attributes(5)`, `pcie(4)`, `npe(7D)`

PCI Express Base Specification v1.0a —2003

Writing Device Drivers

IEEE 1275 PCI Bus Binding — 1998

http://playground.sun.com/1275/bindings/pci/pci-express.txt

Name pcipsy – PCI bus to Safari bus nexus driver

Description The `pcipsy` nexus driver is used for certain IO assemblies for sun4u and high-end Enterprise E10000 servers..

Files /platform/SUNW,Ultra-Enterprise-10000/kernel/drv/sparcv9/pcipsy
32–bit ELF kernel module.

`/platform/sun4u/kernel/drv/sparcv9/pcipsy`
64-bit ELF kernel module.

See Also `pcisch(7D)`

Name pcisch – PCI Bus to Safari bus nexus driver

Description The `pcisch` nexus driver is used for Schizo and XMITS-based I/O assemblies for the following high-end and midrange Sun enterprise servers: Sun Fire E15K, Sun Fire E25K, Sun Fire E2900, Sun Fire E4900 and Sun Fire E6900.

Files `/platform/sun4u/kernel/drv/sparcv9/pcisch` 64-bit ELF kernel module.

See Also `pcipsy`(7D)

Name pckt – STREAMS Packet Mode module

Synopsis

```
int ioctl( fd, I_PUSH, "pckt");
```

Description `pckt` is a STREAMS module that may be used with a pseudo terminal to packetize certain messages. The `pckt` module should be pushed (see `I_PUSH` on `streamio(7I)`) onto the master side of a pseudo terminal.

Packetizing is performed by prefixing a message with an `M_PROTO` message. The original message type is stored in the 1 byte data portion of the `M_PROTO` message.

On the read-side, only the `M_PROTO`, `M_PCPROTO`, `M_STOP`, `M_START`, `M_STOPI`, `M_STARTI`, `M_IOCTL`, `M_DATA`, `M_FLUSH`, and `M_READ` messages are packetized. All other message types are passed upstream unmodified.

Since all unread state information is held in the master's stream head read queue, flushing of this queue is disabled.

On the write-side, all messages are sent down unmodified.

With this module in place, all reads from the master side of the pseudo terminal should be performed with the `getmsg(2)` or `getpmsg`() function. The control part of the message contains the message type. The data part contains the actual data associated with that message type. The onus is on the application to separate the data into its component parts.

See Also `getmsg(2)`, `ioctl(2)`, `ldterm(7M)`, `ptem(7M)`, `streamio(7I)`, `termio(7I)`

STREAMS Programming Guide

Name pcmcia – PCMCIA nexus driver

Description The PCMCIA nexus driver supports PCMCIA card client device drivers. There are no user-configurable options for this driver.

Files `/kernel/misc/pcmcia` `pcmcia` driver

Name pcn – AMD PCnet Ethernet controller device driver

Synopsis `/dev/pcn`

Description The `pcn` Ethernet driver is a multi-threaded, loadable, clonable driver for the AMD PCnet family of Ethernet controllers that use the Generic LAN Driver (`GLD`) facility to implement the required STREAMS and Data Link Provider (see `dlpi(7P)`) interfaces.

This driver supports a number of integrated motherboards and add-in adapters based on the AMD PCnet-ISA, PCnet-PCI, and PCnet-32 controller chips. The `pcn` driver functions include controller initialization, frame transmit and receive, functional addresses, promiscuous and multicast support, and error recovery and reporting.

Application Programming Interface The cloning character-special device, `/dev/pcn`, is used to access all PCnet devices installed in the system.

`pcn` and DLPI The `pcn` driver uses the Solaris `GLD` module which handles all the STREAMS and DLPI specific functions of the driver. It is a *style 2* DLPI driver and therefore supports only the connectionless mode of data transfer. Thus, a DLPI user should issue a `DL_ATTACH_REQ` primitive to select the device to be used. Valid DLPI primitives are defined in `<sys/dlpi.h>`. Refer to `dlpi(7P)` for more information.

The device is initialized on the first attach and de-initialized (stopped) on the last detach.

The values returned by the driver in the `DL_INFO_ACK` primitive in response to a `DL_INFO_REQ` from the user are:

- Maximum SDU is `1500` (`ETHERMTU` - defined in `<sys/ethernet.h>`).
- Minimum SDU is `0`.
- `DLSAP` address length is `8`.
- MAC type is `DL_ETHER`.
- `sap` length value is –2, meaning the physical address component is followed immediately by a 2-byte `sap` component within the `DLSAP` address.
- Service mode is `DL_CLDLS`.
- No optional quality of service (QOS) support is included at present, accordingly, the QOS fields are `0`.
- Provider style is `DL_STYLE2`.
- Version is `DL_VERSION_2`.
- Broadcast address value is the Ethernet/IEEE broadcast address (`FF:FF:FF:FF:FF:FF`).

Once in the `DL_ATTACHED` state, the user must send a `DL_BIND_REQ` to associate a particular Service Access Point (SAP) with the stream.

Known Problems and Limitations

- Occasional data corruption has occurred when `pcn` and `pcscsi` drivers in HP Vectra XU 5/90 and Compaq Deskpro XL systems are used under high network and SCSI loads. These drivers do not perform well in a production server. A possible workaround is to disable the `pcn` device with the system BIOS and use a separate add-in network interface.
- The Solaris `pcn` driver does not support IRQ 4.

Files

`/dev/pcn`	Character special device
`/kernel/drv/pcn.conf`	Configuration file

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86

See Also

`attributes(5)`, `standards(5)`, `dlpi(7P)`, `streamio(7I)`

Writing Device Drivers

STREAMS Programming Guide

Name pcser – PCMCIA serial card device driver

Synopsis

```
serial@<socket>:pcser

serial@<socket>:pcser,cu
```

Description The `pcser` PCMCIA serial card device driver supports asynchronous serial I/O access to any PCMCIA card that that complies with Revision 2.1 of the PCMCIA standard and which represents an 8250-type UART interface.

Preconfigure If a PC card modem or serial device is recognized, the `pcser` device driver is automatically loaded, ports and IRQs allocated, and special files created (if they don't already exist).

Configuration Configuration steps include initial installation and configuration, identifying an unrecognized device and misidentifying a recognized device.

Initial Installation and Configuration

1. Install the Solaris software.
2. Boot the system.
3. Insert the modem or serial device.

Identifying an Unrecognized Device

If you insert a PC card modem or serial device and it is not recognized (that is, no special files are created under `/dev/cua` or `/dev/term`), use the `prtconf` command to find the problem:

1. Become root.
2. Run the `prtconf -D` command to see if your modem or serial device is recognized. An unrecognized device will appear at the end of the `prtconf` output. For example:

```
# prtconf -D
.  .  .
pcic, instance #0 (driver name: pcic)
   .  .  .
    pccard111.222 (driver not attached)
```

3. If your device is not recognized, use the `add_drv` command to add the name of your device as another known alias for `pcser` devices. For example, type the following at the command line:

```
# add_drv -i'"pccard111.222"' pcser
```

Note – Include the double quotes in single quotes to keep the shell from stripping out the double quotes. Use the identification string listed in the `prtconf` output. Use the entire string in the add_drv command. See `add_drv(1M)`.

Misidentifying a Recognized Device

1. Run the `prtconf -D` command to see if your modem or serial device is erroneously recognized as a memory card. If the device is incorrectly recognized as a memory card, the output of the `prtconf` command could show:

```
# prtconf -D

.  .  .
pcic, instance #0 (driver name: pcic)
   .  . .
```

```
memory, instance #0 (driver name: pcmem)
pcram, instance #0 (driver name: pcram)
```

2. Use the Configuration Assistant to identify the memory resource conflict, and add correct information for the device on the View/Edit Devices menu. Typically, the problem may be a resource conflict between device memory settings. A PC Card adapter chip that is not fully supported may also be the cause of the problem.
3. To work properly with the Solaris operating environment, all devices must be accounted for, even those the Solaris environment does not support. The Configuration Assistant software accounts for all devices in your system.

Additional Configuration When adding a new serial port or modem to the system, you often need to edit configuration files so that applications can use the new communications port. For example, the `/etc/uucp/Devices` file needs to be updated to use UUCP. See *Overview of UUCP* in the *System Administration Guide*. For PPP on the serial port, see `pppd`(1M) and *Solaris PPP Overview* in the *System Administration Guide*.

Special Files The serial devices in `/dev/term` and `/dev/cua` are named by socket number. A card inserted in socket 0 is `pc0`, and socket 1 is `pc1`.

Hot Plugging If a PC Card modem or serial device is unplugged while in use, the device driver returns errors until the card is replaced in the socket.

The device must be closed and reopened with the card reinserted before the device begins working again. The restart process depends on the application. For example, a `tip` session automatically exits when a card in use is unplugged. To restart the system, you must restart the `tip` session.

Files

`/kernel/drv/pcser`	`pcser` driver
`/dev/term/pc`*n*	dial-in devices
`/dev/cua/pc`*n*	dial-out devices where: *n* is the PCMCIA physical socket number.

See Also `cu`(1C), `tip`(1), `uucp`(1C), `autopush`(1M), `ports`(1M), `ioctl`(2), `open`(2), `pcmcia`(7D), `termio`(7I), `ldterm`(7M), `ttcompat`(7M)

Diagnostics

`pcser: socket`*n* `soft silo overflow`
The driver's character input ring buffer overflowed before it could be serviced.

`pcser: socket`*n* `unable to get CIS information`
The CIS on the card has incorrect information or is in an incorrect format. This message usually indicates a non-compliant card.

Name pcwl – Lucent/PrismII 802.11b wireless NIC driver

Description The `pcwl` *802.11b* wireless NIC driver is a multi- threaded, loadable, clonable, GLDv3-based STREAMS driver. It supports the pccard and PCI/MiniPCI cards with the Lucent and PrismII *802.11b* chipsets on x86 and SPARC.

Driver Configuration The `pcwl` driver supports *802.11b* data rates of 1, 2, 5.5 and 11 (Mbits/sec). The default is 11.

The `pcwl` driver supports BSS networks (also known as "ap" or "infrastructure" networks) and IBSS (or "ad-hoc") networks. For authentication type, the `pcwl` driver supports the "open" (or "open-system") mode and the "shared-key" mode. For encryption type, only WEP is currently supported. You perform configuration and administration tasks using the `dladm(1M)` and `wificonfig(1M)` utilities.

Files

`/dev/pcwl*`	Special character device.
/kernel/drv/pcwl	32-bit ELF kernel module (x86).
`/kernel/drv/amd64/pcwl`	64-bit ELF kernel module (x86).
`/kernel/drv/sparcv9/pcwl`	64-bit ELF kernel module (SPARC).

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SPARC, x86

See Also `dladm(1M)`, `wificonfig(1M)`, `attributes(5)`, `gld(7D)`

802.11b Standard for Wireless Local Area Networks (WLANs) - IEEE

Name pfb – Sun XVR-50 and XVR-100 Graphics Accelerator device driver

Description The `pfb` driver is the device driver for the Sun XVR-50 and XVR-100 Graphics Accelerator.

Files

`/dev/fbs/pfb\fIn`	Device special file for XVR-50 or XVR-100 single screen.
`/dev/fbs/pfb\fIn\fPa`	Device special file for the XVR-100 first video out.
`/dev/fbs/pfb\fIn\fPb`	Device special file for the XVR-100 second video out.

See Also `SUNWpfb_config(1M)`

Name pf_key – Security association database interface

Synopsis

```
#include <sys/types.h>
#include <sys/socket.h>
#include <net/pfkeyv2.h>


int socket(PF_KEY,SOCK_RAW,PF_KEY_V2);
```

Description Keying information for IPsec security services is maintained in security association databases (SADBs). The security associations (SAs) are used to protect both inbound and outbound packets.

A user process (or possibly multiple co-operating processes) maintains SADBs by sending messages over a special kind of socket. This is analogous to the method described in route(7P). Only a superuser may access an SADB.

SunOS applications that use PF_KEY include ipseckey(1M) and in.iked(1M).

The operating system can spontaneously send pf_key messages to listening processes, such as a request for a new SA for an outbound datagram or to report the expiration of an existing SA.

One opens the channel for passing SADB control messages by using the socket call shown in the Synopsis section above. More than one key socket can be open per system.

Messages are formed by a small base header, followed by zero or more extension messages, some of which require additional data following them. The base message and all extensions must be eight-byte aligned. An example message is the GET message, which requires the base header, the SA extension, and the ADDRESS_DST extension.

Messages Messages include:

```
#define    SADB_GETSPI  /* Get a new SPI value from the system. */
#define    SADB_UPDATE     /* Update an SA. */
#define    SADB_ADD        /* Add a fully-formed SA. */
#define    SADB_DELETE     /* Delete an SA. */
#define    SADB_GET        /* Get an SA */
#define    SADB_ACQUIRE    /* Kernel needs a new SA. */
#define    SADB_REGISTER   /* Regis. to receive ACQUIRE msgs. */
#define    SADB_EXPIRE     /* SA has expired. */
#define    SADB_FLUSH      /* Flush all SAs. */
#define    SADB_DUMP       /* Get all SAs.  (Unreliable) */
#define    SADB_X_PROMISC  /* Listen promiscuously */
#define    SADB_X_INVERSE_ACQUIRE  /* Query kernel policy,
                                      get an ACQUIRE in return. */
#define    SADB_X_UPDATEPAIR     /* Update an SA and its pair SA */
#define    SADB_X_DELPAIR        /* Delete an SA pair. */
```

The base message header consists of:

```
struct sadb_msg {
    uint8_t  sadb_msg_version;  /* Set to PF_KEY_V2, for compat. */
    uint8_t  sadb_msg_type;     /* Msg. type */
    uint8_t  sadb_msg_errno;    /* Why message failed */
    uint8_t  sadb_msg_satype;   /* Which security service */
    uint16_t sadb_msg_len;      /* Length in 8-byte units */
    uint16_t sadb_msg_reserved; /* Zero out */
 #define sadb_x_msg_diagnostic sadb_msg_reserved
                               /* Extended diagnostics for errors */
    uint32_t sadb_msg_seq;      /* For msg. originator */
    uint32_t sadb_msg_pid;      /* ID originator */
};
```

Extension types include:

```
#define  SADB_EXT_SA                /* SA info */
#define  SADB_EXT_LIFETIME_HARD     /* Hard lifetime */
#define  SADB_EXT_LIFETIME_SOFT     /* Soft lifetime */
#define  SADB_EXT_ADDRESS_SRC       /* Source address */
#define  SADB_EXT_ADDRESS_DST       /* Destination address */
#define  SADB_EXT_ADDRESS_PROXY     /* Proxy address - DEPRECATED */
#define  SADB_EXT_KEY_AUTH          /* Authen. key */
#define  SADB_EXT_KEY_ENCRYPT       /* Encryption key */
#define  SADB_EXT_IDENTITY_SRC      /* Source certif. ID */
#define  SADB_EXT_IDENTITY_DST      /* Destination certif. ID */
#define  SADB_EXT_SENSITIVITY       /* Sensitivity info */
#define  SADB_EXT_PROPOSAL          /* Security proposal */
#define  SADB_EXT_SUPPORTED_AUTH   /* Supported authen. algo's */
#define  SADB_EXT_SUPPORTED_ENCRYPT /* Supported encryption algo's */
#define  SADB_EXT_SPIRANGE          /* Range of possible SPIs *
#define  SADB_X_EXT_EREG           /* Reg. for extended ACQUIRE */
#define  SADB_X_EXT_EPROP          /* Extended ACQUIRE proposals */
#define  SADB_X_EXT_KM_COOKIE      /* Indicates which KM derived SA. */
#define  SADB_X_EXT_ADDRESS_NATT_LOC  /* NAT-Traversal local (my public) */
#define  SADB_X_EXT_ADDRESS_NATT_REM  /* NAT-T remote (peer's private) */
#define  SADB_X_EXT_ADDRESS_INNER_SRC /* Tunnel-mode inner source */
#define  SADB_X_EXT_ADDRESS_INNER_DST /* Tunnel-mode inner dest */
#define  SADB_X_EXT_REPLAY_VALUE      /* Replay Value */
#define  SADB_X_EXT_LIFETIME_IDLE     /* Idle lifetime */
#define  SADB_X_EXT_PAIR              /* SA pair extension*/
#define  SADB_X_EXT_OUTER_SENS   /*outer sensitivity */
```

Security Association Information Extension flags:

```
#define SADB_SAFLAGS_PFS        0x1          /* Perfect forward secrecy? */
#define SADB_SAFLAGS_NOREPLAY   0x2          /* Replay field NOT PRESENT. */
#define SADB_X_SAFLAGS_USED     0x80000000   /* SA used/not used */
```

```
#define SADB_X_SAFLAGS_UNIQUE   0x40000000   /* SA unique/reusable */
#define SADB_X_SAFLAGS_AALG1    0x20000000   /* Auth-alg specif. flag 1 */
#define SADB_X_SAFLAGS_AALG2    0x10000000   /* Auth-alg specif. flag 2 */
#define SADB_X_SAFLAGS_EALG1     0x8000000   /* Encr-alg specif. flag 1 */
#define SADB_X_SAFLAGS_EALG2     0x4000000   /* Encr-alg specif. flag 2 */
#define SADB_X_SAFLAGS_KM1       0x2000000   /* Key mgmt. specif. flag 1 */
#define SADB_X_SAFLAGS_KM2       0x1000000   /* Key mgmt. specif. flag 2 */
#define SADB_X_SAFLAGS_KM3        0x800000   /* Key mgmt. specif. flag 3 */
#define SADB_X_SAFLAGS_KM4        0x400000   /* Key mgmt. specif. flag 4 */
#define SADB_X_SAFLAGS_KRES1      0x200000   /* Reserved by the kernel */
#define SADB_X_SAFLAGS_NATT_LOC   0x100000   /* this has a natted srcSA */
#define SADB_X_SAFLAGS_NATT_REM    0x80000   /* this has a natted dstSA */
#define SADB_X_SAFLAGS_KRES2       0x40000   /* Reserved by the kernel */
#define SADB_X_SAFLAGS_TUNNEL      0x20000   /* tunnel mode */
#define SADB_X_SAFLAGS_PAIRED      0x10000   /* inbound/outbound pair*/
#define SADB_X_SAFLAGS_OUTBOUND     0x8000   /* SA direction bit */
#define SADB_X_SAFLAGS_INBOUND      0x4000   /* SA direction bit */
```

Sensitivity Extension flags:

```
#define SADB_X_SENS_IMPLICIT 0x1    /* implicit labelling */
#define SADB_X_SENS_UNLABELED 0x2   /* peer is unlabeled */
```

Extension headers include:

Generic Extension Header

```
struct sadb_ext {
    uint16_t sadb_ext_len;  /* In 64-bit words, inclusive */
    uint16_t sadb_ext_type; /* 0 is reserved */
};
```

Security Association Information Extension

```
struct sadb_sa {
    uint16_t sadb_sa_len;
    uint16_t sadb_sa_exttype; /* ASSOCIATION */
    uint32_t sadb_sa_spi;
    uint8_t sadb_sa_replay;
    uint8_t sadb_sa_state;
    uint8_t sadb_sa_auth;
    uint8_t sadb_sa_encrypt;
    uint32_t sadb_sa_flags;
};
```

Lifetime Extension

```
struct sadb_lifetime {
    uint16_t sadb_lifetime_len;
    uint16_t sadb_lifetime_exttype;   /* SOFT, HARD, CURRENT */
```

```
    uint32_t sadb_lifetime_allocations;
    uint64_t sadb_lifetime_bytes;
    uint64_t sadb_lifetime_addtime;
    uint64_t sadb_lifetime_usetime;
};
```

Address Extension

```
struct sadb_address {
    uint16_t sadb_address_len;
    uint16_t sadb_address_exttype;    /* SRC, DST, NATT_*, INNER_* */
    uint8_t sadb_address_proto;       /* Proto for ports... */
    uint8_t sadb_address_prefixlen;   /* Prefix length for INNER_*. */
    uint16_t sadb_address_reserved;   /* Padding */
                                      /* Followed by a sockaddr
                                         structure.*/
};
```

Keying Material Extension

```
struct sadb_key {
    uint16_t sadb_key_len;
    uint16_t sadb_key_exttype;         /* AUTH, ENCRYPT */
    uint16_t sadb_key_bits;
    uint16_t sadb_key_reserved;
        /* Followed by actual key(s) in
         canonical (outbound proc.) order. */
};
```

Indentity Extension

```
struct sadb_ident {
    uint16_t sadb_ident_len;
    uint16_t sadb_ident_exttype;      /* SRC, DST, PROXY */
    uint16_t sadb_ident_type;         /* FQDN, USER_FQDN, etc. */
    uint16_t sadb_ident_reserved;     /* Padding */
    uint64_t sadb_ident_id;           /* For userid, etc. */
        /* Followed by an identity null-terminate C string if present. */
};
```

Sensitivity/Integrity Extension

```
struct sadb_sens {
  uint16_t sadb_sens_len;
  uint16_t sadb_sens_exttype; /* SENSITIVITY, OUTER_SENS */
  uint32_t sadb_sens_dpd;
  uint8_t sadb_sens_sens_level;
  uint8_t sadb_sens_sens_len;   /* 64-bit words */
  uint8_t sadb_sens_integ_level;
  uint8_t sadb_sens_integ_len;  /* 64-bit words */
  uint32_t sadb_x_sens_flags;
```

```
                    /*
                     * followed by two uint64_t arrays
                     * uint64_t sadb_sens_bitmap[sens_bitmap_len];
                     * uint64_t integ_bitmap[integ_bitmap_len];
                     */
};
```

Proposal Extension

```
struct sadb_prop {
    uint16_t sadb_prop_len;
    uint16_t sadb_prop_exttype;     /* PROPOSAL, X_EPROP */
    uint8_t sadb_prop_replay;
    uint8_t sadb_X_prop_ereserved;
    uint16_t sadb_x_prop_numecombs;
/* Followed by sadb_comb[] array or sadb_ecomb[] array. */
};
```

Combination Instance for a Proposal

```
struct sadb_comb {
    uint8_t sadb_comb_auth;
    uint8_t sadb_comb_encrypt;
    uint16_t sadb_comb_flags;
    uint16_t sadb_comb_auth_minbits;
    uint16_t sadb_comb_auth_maxbits;
    uint16_t sadb_comb_encrypt_minbits;
    uint16_t sadb_comb_encrypt_maxbits;
    uint32_t sadb_comb_reserved;
    uint32_t sadb_comb_soft_allocations;
    uint32_t sadb_comb_hard_allocations;
    uint64_t sadb_comb_soft_bytes;
    uint64_t sadb_comb_hard_bytes;
    uint64_t sadb_comb_soft_addtime;
    uint64_t sadb_comb_hard_addtime;
    uint64_t sadb_comb_soft_usetime;
    uint64_t sadb_comb_hard_usetime;
};
```

Extended Combination

```
struct sadb_x_ecomb {
    uint8_t sadb_x_ecomb_numalgs;
    uint8_t sadb_x_ecomb_reserved;
    uint16_t sadb_x_ecomb_flags;    /* E.g. PFS? */
    uint32_t sadb_x_ecomb_reserved2;
    uint32_t sadb_x_ecomb_soft_allocations;
    uint32_t sadb_x_ecomb_hard_allocations;
    uint64_t sadb_x_ecomb_soft_bytes;
    uint64_t sadb_x_ecomb_hard_bytes;
```

```
    uint64_t sadb_x_ecomb_soft_addtime;
    uint64_t sadb_x_ecomb_hard_addtime;
    uint64_t sadb_x_ecomb_soft_usetime;
    uint64_t sadb_x_ecomb_hard_usetime;
};
```

Extended Combination Algorithm Descriptors

```
struct sadb_x_algdesc {
    uint8_t sadb_x_algdesc_satype;  /* ESP, AH, etc. */
    uint8_t sadb_x_algdesc_algtype; /* AUTH, CRYPT, COMPRESS */
    uint8_t sadb_x_algdesc_alg;     /* DES, 3DES, MD5, etc. */
    uint8_t sadb_x_algdesc_reserved;
    uint16_t sadb_x_algdesc_minbits;        /* Bit strengths. */
    uint16_t sadb_x_algdesc_maxbits;
   };
```

Extended Register

```
struct sadb_x_ereg {
    uint16_t sadb_x_ereg_len;
    uint16_t sadb_x_ereg_exttype;   /* X_EREG */
    uint8_t sadb_x_ereg_satypes[4]; /* Array of SA types, 0-terminated.
|};
```

Key Management Cookie

```
struct sadb_x_kmc {
    uint16_t sadb_x_kmc_len;
    uint16_t sadb_x_kmc_exttype;    /* X_KM_COOKIE */
    uint32_t sadb_x_kmc_proto;      /* KM protocol */
    uint32_t sadb_x_kmc_cookie;     /* KMP-specific */
    uint32_t sadb_x_kmc_reserved;   /* Reserved; must be zero */
   };
```

Supported Algorithms Extension

```
struct sadb_supported {
    uint16_t sadb_supported_len;
    uint16_t sadb_supported_exttype;
    uint32_t sadb_supported_reserved;
};
```

Algorithm Instance

```
struct sadb_alg {
    uint8_t sadb_alg_id;        /* Algorithm type. */
    uint8_t sadb_alg_ivlen;     /* IV len, in bits */
    uint16_t sadb_alg_minbits;  /* Min. key len (in bits) */
    uint16_t sadb_alg_maxbits;  /* Max. key length */
    uint16_t sadb_alg_reserved;
};
```

SPI Extension Range

```
struct sadb_spirange {
    uint16_t sadb_spirange_len;
    uint16_t sadb_spirange_exttype;    /* SPI_RANGE */
    uint32_t sadb_spirange_min
    uint32_t sadb_spirange_max;
    uint32_t sadb_spirange_reserved;
};
```

Security Association Pair Extension

```
struct sadb_x_pair {
    uint16_t sadb_x_pair_len;
    uint16_t sadb_x_pair_exttype;   /* SADB_X_EXT_PAIR */
    uint32_t sadb_x_pair_spi;       /* SPI of paired SA */
};
```

Replay Value

```
struct sadb_x_replay_ctr {
    uint16_t sadb_x_rc_len;
    uint16_t sadb_x_rc_exttype;
    uint32_t sadb_x_rc_replay32;    /* For 240x SAs. */
    uint64_t sadb_x_rc_replay64;    /* For 430x SAs. */
};
```

Message Use and Behavior

Each message has a behavior. A behavior is defined as where the initial message travels, for example, user to kernel, and what subsequent actions are expected to take place. Contents of messages are illustrated as:

```
<base, REQUIRED EXTENSION, REQ., (OPTIONAL EXTENSION,) (OPT)>
```

The SA extension is sometimes used only for its SPI field. If all other fields must be ignored, this is represented by SA(*).

The lifetime extensions are represented with one to three letters after the word lifetime, representing (H)ARD, (S)OFT, and (C)URRENT.

The address extensions are represented with one to three letters after the word `address`, representing (S)RC, (D)ST, (Nl)NAT-T local, (Nr)NAT-T remote, (Is)Inner source, and (Id)Inner destination.

Source and destination address extensions reflect outer-header selectors for an IPsec SA. An SA is inbound or outbound depending on which of the source or destination address is local to the node. Inner-source and inner-destination selectors represent inner-header selectors for Tunnel Mode SAs. A Tunnel Mode SA *must* have either IPPROTO_ENCAP or IPPROTO_IPV6 in its outer-headers as protocol selector, in addition to filled-in Inner-address extensions.

NAT-T local and NAT-T remote addresses store local and remote ports used for ESP-in-UDP encapsulation. A non-zero local NAT-T address extension represents the local node's external IP address if it is not equivalent to the SA's local address. A non-zero remote NAT-T address represents a peer's behind-a-NAT address if it is not equivalent to the SA's remote address. An SA with NAT-T extensions protects-and-transmits outbound traffic. Processing of inbound NAT-T traffic requires a UDP socket bound to the appropriate local port and it *must* have the UDP_NAT_T_ENDPOINT (see udp(7P)) socket option enabled.

Note that when an error occurs, only the base header is sent. In the event of an error, an extended diagnostic can be set (see DIAGNOSTICS). Typical errors include:

`EINVAL` Various message improprieties, including SPI ranges that are malformed, weak keys, and others. If EINVAL is returned, an application should look at the `sadb_x_msg_diagnostic` field of the sadb_msg structure. It contains one of many possible causes for EINVAL. See `net/pfkeyv2.h` for values, all of the form SADB_X_DIAGNOSTIC_.

`ENOMEM` Needed memory was not available.

`ENSGSIZ` Message exceeds the maximum length allowed.

`EEXIST` SA (that is being added or created with `GETSPI`) already exists.

`ESRCH` SA could not be found.

The following are examples of message use and behavior:

`SADB_GETSPI`

Send a `SADB_GETSPI` message from a user process to the kernel.

```
<base, address, SPI range>
```

The kernel returns the `SADB_GETSPI` message to all listening processes.

```
<base, SA(*), address (SD)>
```

`SADB_UPDATE`

Send a `SADB_UPDATE` message from a user process to the kernel.

```
<base, SA, (lifetime(HS),) address(SD), (address(Is,Id),
 address(Nl,Nr),key (AE), (identity(SD),) (sensitivity, outer sensitivity)>
```

The kernel returns the `SADB_UPDATE` message to all listening processes.

```
<base, SA(*), address (SD), (pair)>
```

Adding a `sadb_x_pair` extension to an `SADB_UPDATE` or `SADB_ADD` message updates the security association pair linkage with the SPI of the security association contained in that

extension. The resulting security association *pair* can be updated or as a single entity using the SADB_X_UPDATEPAIR or SADB_X_DELPAIR message types.

SADB_ADD

Send a SADB_ADD message from a user process to the kernel.

```
<base, SA, (lifetime(HS),) address(SD), (address(Is,Id),)
  (address(Nl,Nr),) key(AE), (identity(SD),) (sensitivity, outer sensitivity) (pair)>
```

The kernel returns the SADB_ADD message to all listening processes.

```
<base, SA, (lifetime(HS),) address (SD), (address(Is,Id),)
   (address(Nl,Nr),) (identity (SD),) (sensitivity, outer sensitivity)>
```

SADB_X_UPDATEPAIR

Send a SADB_X_UPDATEPAIR message from a user process to the kernel. This message type is used to update the lifetime values of a security association and the lifetime values of the security association it is paired with.

```
<base, SA, lifetime(HS), address(SD)>
```

SADB_DELETE | SADB_X_DELPAIR

Send a SADB_DELETE message from a user process to the kernel. The SADB_X_DELPAIR message type requests deletion of the security association and the security association it is paired with.

```
<base, SA (*), address (SD)>
```

The kernel returns the SADB_DELETE message to all listening processes.

```
<base, SA (*), address (SD)>
```

SADB_GET

Send a SADB_GET message from a user process to the kernel.

```
<base, SA (*), address (SD)>
```

The kernel returns the SADB_GET message to the socket that sent the SADB_GET message.

```
<base, SA , (lifetime (HSC),) address SD), (address (P),) key (AE),
   (identity (SD),) (sensitivity, outer sensitivity)>
```

SADB_ACQUIRE

The kernel sends a SADB_ACQUIRE message to registered sockets. Note that any GETSPI, ADD, or UPDATE calls in reaction to an ACQUIRE must fill in the sadb_msg_seq of those messages with the one in the ACQUIRE message. The address (SD) extensions must have the port fields filled in with the port numbers of the session requiring keys if appropriate.

```
<base, address (SD), (address(Is,Id)), (identity(SD),)
     (sensitivity) proposal>
```

Extended ACQUIRE has a slightly different format. The `sadb_msg_satype` field is 0, and the extension contains the desired combination(s) of security protocols.

```
<base, address (SD), (address(Is,Id)), (identity(SD),)
                       (sensitivity,) eprop>
```

If key management fails, send an `SADB_ACQUIRE` to indicate failure.

```
<base>
```

`SADB_X_INVERSE_ACQUIRE`

For inbound Key Management processing, a Key Management application can wish to consult the kernel for its policy. The application should send to the kernel:

```
<base, address (SD), (address(Is,Id))>
```

The kernel returns a message similar to a kernel-generated extended ACQUIRE:

```
<base, address (SD), (address(Is,Id)), (identity(SD),)
                       (sensitivity,) eprop>
```

`SADB_REGISTER`

Send a `SADB_REGISTER` message from a user process to the kernel.

```
<base>
```

The kernel returns the `SADB_REGISTER` message to registered sockets, with algorithm types supported by the kernel being indicated in the supported algorithms field. Note that this message can arrive asynchronously due to an algorithm being loaded or unloaded into a dynamically linked kernel.

```
<base, supported>
```

There is also the extended REGISTER, which allows this process to receive extended ACQUIREs.

```
<base, ereg>
```

Which returns a series of SADB_REGISTER replies (one for each security protocol registered) from the kernel.

`SADB_EXPIRE`

The kernel sends a `SADB_EXPIRE` message to all listeners when the soft limit of a security association has been expired.

```
<base, SA, lifetime (C and one of HS), address (SD)>
```

SADB_FLUSH

Send a SADB_FLUSH message from a user process to the kernel.

<base>

The kernel returns the SADB_FLUSH message to all listening sockets.

<base>

SADB_DUMP

Send a SADB_DUMP message from a user process to the kernel.

<base>

Several SADB_DUMP messages returns from the kernel to the sending socket.

```
<base, SA, (lifetime (HSC),) address (SD), (address (Is,Id),)
   (address (Nl,Nr),) key (AE), (identity (SD),) sensitivity, outer sensitivity)>
```

To mark the end of a dump a single base header arrives with its sadb_mdg_seq set to 0.

<base>

SADB_X_PROMISC

Send a SADB_X_PROMISC message from a user process to the kernel.

<base>

The kernel returns the SADB_X_PROMISC message to all listening processes.

<base>

Diagnostics The message returning from the kernel contains a diagnostic value in the base message header, the diagnostic value indicates if action requested by the original message was a success.

Diagnostic Values:

```
#define SADB_X_DIAGNOSTIC_NONE                  0
#define SADB_X_DIAGNOSTIC_UNKNOWN_MSG           1
#define SADB_X_DIAGNOSTIC_UNKNOWN_EXT           2
#define SADB_X_DIAGNOSTIC_BAD_EXTLEN            3
#define SADB_X_DIAGNOSTIC_UNKNOWN_SATYPE        4
#define SADB_X_DIAGNOSTIC_SATYPE_NEEDED         5
#define SADB_X_DIAGNOSTIC_NO_SADBS              6
#define SADB_X_DIAGNOSTIC_NO_EXT                7
        /* Bad address family value */
#define SADB_X_DIAGNOSTIC_BAD_SRC_AF            8
        /* in sockaddr->sa_family. */
#define SADB_X_DIAGNOSTIC_BAD_DST_AF            9
        /* These two are synonyms. */
```

```
#define SADB_X_DIAGNOSTIC_BAD_PROXY_AF          10
#define SADB_X_DIAGNOSTIC_BAD_INNER_SRC_AF      10

#define SADB_X_DIAGNOSTIC_AF_MISMATCH           11

#define SADB_X_DIAGNOSTIC_BAD_SRC               12
#define SADB_X_DIAGNOSTIC_BAD_DST               13

#define SADB_X_DIAGNOSTIC_ALLOC_HSERR           14
#define SADB_X_DIAGNOSTIC_BYTES_HSERR           15
#define SADB_X_DIAGNOSTIC_ADDTIME_HSERR         16
#define SADB_X_DIAGNOSTIC_USETIME_HSERR         17

#define SADB_X_DIAGNOSTIC_MISSING_SRC           18
#define SADB_X_DIAGNOSTIC_MISSING_DST           19
#define SADB_X_DIAGNOSTIC_MISSING_SA            20
#define SADB_X_DIAGNOSTIC_MISSING_EKEY          21
#define SADB_X_DIAGNOSTIC_MISSING_AKEY          22
#define SADB_X_DIAGNOSTIC_MISSING_RANGE         23

#define SADB_X_DIAGNOSTIC_DUPLICATE_SRC         24
#define SADB_X_DIAGNOSTIC_DUPLICATE_DST         25
#define SADB_X_DIAGNOSTIC_DUPLICATE_SA          26
#define SADB_X_DIAGNOSTIC_DUPLICATE_EKEY        27
#define SADB_X_DIAGNOSTIC_DUPLICATE_AKEY        28
#define SADB_X_DIAGNOSTIC_DUPLICATE_RANGE       29

#define SADB_X_DIAGNOSTIC_MALFORMED_SRC         30
#define SADB_X_DIAGNOSTIC_MALFORMED_DST         31
#define SADB_X_DIAGNOSTIC_MALFORMED_SA          32
#define SADB_X_DIAGNOSTIC_MALFORMED_EKEY        33
#define SADB_X_DIAGNOSTIC_MALFORMED_AKEY        34
#define SADB_X_DIAGNOSTIC_MALFORMED_RANGE       35

#define SADB_X_DIAGNOSTIC_AKEY_PRESENT          36
#define SADB_X_DIAGNOSTIC_EKEY_PRESENT          37
#define SADB_X_DIAGNOSTIC_PROP_PRESENT          38
#define SADB_X_DIAGNOSTIC_SUPP_PRESENT          39
#define SADB_X_DIAGNOSTIC_BAD_AALG              40
#define SADB_X_DIAGNOSTIC_BAD_EALG              41
#define SADB_X_DIAGNOSTIC_BAD_SAFLAGS           42
#define SADB_X_DIAGNOSTIC_BAD_SASTATE           43

#define SADB_X_DIAGNOSTIC_BAD_AKEYBITS          44
#define SADB_X_DIAGNOSTIC_BAD_EKEYBITS          45

#define SADB_X_DIAGNOSTIC_ENCR_NOTSUPP          46
```

```
#define SADB_X_DIAGNOSTIC_WEAK_EKEY             47
#define SADB_X_DIAGNOSTIC_WEAK_AKEY             48

#define SADB_X_DIAGNOSTIC_DUPLICATE_KMP         49
#define SADB_X_DIAGNOSTIC_DUPLICATE_KMC         50

#define SADB_X_DIAGNOSTIC_MISSING_NATT_LOC      51
#define SADB_X_DIAGNOSTIC_MISSING_NATT_REM      52
#define SADB_X_DIAGNOSTIC_DUPLICATE_NATT_LOC    53
#define SADB_X_DIAGNOSTIC_DUPLICATE_NATT_REM    54
#define SADB_X_DIAGNOSTIC_MALFORMED_NATT_LOC    55
#define SADB_X_DIAGNOSTIC_MALFORMED_NATT_REM    56
#define SADB_X_DIAGNOSTIC_DUPLICATE_NATT_PORTS  57

#define SADB_X_DIAGNOSTIC_MISSING_INNER_SRC     58
#define SADB_X_DIAGNOSTIC_MISSING_INNER_DST     59
#define SADB_X_DIAGNOSTIC_DUPLICATE_INNER_SRC   60
#define SADB_X_DIAGNOSTIC_DUPLICATE_INNER_DST   61
#define SADB_X_DIAGNOSTIC_MALFORMED_INNER_SRC   62
#define SADB_X_DIAGNOSTIC_MALFORMED_INNER_DST   63

#define SADB_X_DIAGNOSTIC_PREFIX_INNER_SRC      64
#define SADB_X_DIAGNOSTIC_PREFIX_INNER_DST      65
#define SADB_X_DIAGNOSTIC_BAD_INNER_DST_AF      66
#define SADB_X_DIAGNOSTIC_INNER_AF_MISMATCH     67

#define SADB_X_DIAGNOSTIC_BAD_NATT_REM_AF       68
#define SADB_X_DIAGNOSTIC_BAD_NATT_LOC_AF       69

#define SADB_X_DIAGNOSTIC_PROTO_MISMATCH        70
#define SADB_X_DIAGNOSTIC_INNER_PROTO_MISMATCH  71

#define SADB_X_DIAGNOSTIC_DUAL_PORT_SETS        72

#define SADB_X_DIAGNOSTIC_PAIR_INAPPROPRIATE    73
#define SADB_X_DIAGNOSTIC_PAIR_ADD_MISMATCH     74
#define SADB_X_DIAGNOSTIC_PAIR_ALREADY          75
#define SADB_X_DIAGNOSTIC_PAIR_SA_NOTFOUND      76
#define SADB_X_DIAGNOSTIC_BAD_SA_DIRECTION      77

#define SADB_X_DIAGNOSTIC_SA_NOTFOUND           78
#define SADB_X_DIAGNOSTIC_SA_EXPIRED            79
#define SADB_X_DIAGNOSTIC_BAD_CTX               80
#define SADB_X_DIAGNOSTIC_INVALID_REPLAY        81
#define SADB_X_DIAGNOSTIC_MISSING_LIFETIME      82
#define SADB_X_DIAGNOSTIC_BAD_LABEL             83
```

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsr
Interface Stability	Committed

See Also `in.iked(1M)`, `ipseckey(1M)`, `ipsec(7P)`, `ipsecah(7P)`, `ipsecesp(7P)`, `route(7P)`, `udp(7P)`

McDonald, D.L., Metz, C.W., and Phan, B.G., *RFC 2367, PF_KEY Key Management API, Version 2*, The Internet Society, July 1998.

Notes Time-based lifetimes might not expire with exact precision in seconds because kernel load can affect the aging of SA's.

Name pfmod – STREAMS Packet Filter Module

Synopsis

```
#include <sys/pfmod.h>

ioctl(fd, IPUSH, "pfmod");
```

Description `pfmod` is a STREAMS module that subjects messages arriving on its read queue to a packet filter and passes only those messages that the filter accepts on to its upstream neighbor. Such filtering can be very useful for user-level protocol implementations and for networking monitoring programs that wish to view only specific types of events.

Read-side Behavior `pfmod` applies the current packet filter to all `M_DATA` and `M_PROTO` messages arriving on its read queue. The module prepares these messages for examination by first skipping over all leading `M_PROTO` message blocks to arrive at the beginning of the message's data portion. If there is no data portion, `pfmod` accepts the message and passes it along to its upstream neighbor. Otherwise, the module ensures that the part of the message's data that the packet filter might examine lies in contiguous memory, calling the `pullupmsg(9F)` utility routine if necessary to force contiguity. (Note: this action destroys any sharing relationships that the subject message might have had with other messages.) Finally, it applies the packet filter to the message's data, passing the entire message upstream to the next module if the filter accepts, and discarding the message otherwise. See PACKET FILTERS below for details on how the filter works.

If there is no packet filter yet in effect, the module acts as if the filter exists but does nothing, implying that all incoming messages are accepted. The ioctls section below describes how to associate a packet filter with an instance of `pfmod`.

`pfmod` passes all other messages through unaltered to its upper neighbor.

Write-side Behavior `pfmod` intercepts `M_IOCTL` messages for the *ioctl* described below. The module passes all other messages through unaltered to its lower neighbor.

ioctls `pfmod` responds to the following *ioctl*.

`PFIOCSETF` This *ioctl* directs the module to replace its current packet filter, if any, with the filter specified by the `struct packetfilt` pointer named by its final argument. This structure is defined in `<sys/pfmod.h>` as:

```
struct packetfilt {
 uchar_t    Pf_Priority;            /* priority of filter */
 uchar_t    Pf_FilterLen;           /* length of filter cmd list */
 ushort_t    Pf_Filter[ENMAXFILTERS]; /* filter command list */
};
```

The `Pf_Priority` field is included only for compatibility with other packet filter implementations and is otherwise ignored. The packet filter itself is specified in the `Pf_Filter` array as a sequence of two-byte commands, with the `Pf_FilterLen` field giving the number of commands in the sequence. This implementation restricts the maximum number of commands in a filter (`ENMAXFILTERS`) to 255. The next section describes the available commands and their semantics.

Packet Filters A packet filter consists of the filter command list length (in units of `ushort_ts`), and the filter command list itself. (The priority field mentioned above is ignored in this implementation.) Each filter command list specifies a sequence of actions that operate on an internal stack of ushort_ts ("shortwords") or an offset register. The offset register is initially zero. Each shortword of the command list specifies an action and a binary operator. Using _n_ as shorthand for the next shortword of the instruction stream and _%oreg_ for the offset register, the list of actions is:

```
COMMAND        SHORTWORDS  ACTION
ENF_PUSHLIT             2  Push _n_ on the stack.
ENF_PUSHZERO            1  Push zero on the stack.
ENF_PUSHONE             1  Push one on the stack.
ENF_PUSHFFFF            1  Push 0xFFFF on the stack.
ENF_PUSHFF00            1  Push 0xFF00 on the stack.
ENF_PUSH00FF            1  Push 0x00FF on the stack.
ENF_LOAD_OFFSET         2  Load _n_ into _%oreg_.
ENF_BRTR                2  Branch forward _n_ shortwords if
                           the top element of the stack is
                           non-zero.
ENF_BRFL                2  Branch forward _n_ shortwords if
                           the top element of the stack is zero.
ENF_POP                 1  Pop the top element from the stack.
ENF_PUSHWORD+m          1  Push the value of shortword (_m_ +
                           _%oreg_) of the packet onto the stack.
```

The binary operators can be from the set {`ENF_EQ`, `ENF_NEQ`, `ENF_LT`, `ENF_LE`, `ENF_GT`,`ENF_GE`, `ENF_AND`, `ENF_OR`, `ENF_XOR`} which operate on the top two elements of the stack and replace them with its result.

When both an action and operator are specified in the same shortword, the action is performed followed by the operation.

The binary operator can also be from the set {`ENF_COR`, `ENF_CAND`, `ENF_CNOR`, `ENF_CNAND`}. These are “short-circuit” operators, in that they terminate the execution of the filter immediately if the condition they are checking for is found, and continue otherwise. All pop two elements from the stack and compare them for equality; `ENF_CAND` returns false if the result is false; `ENF_COR` returns true if the result is true; `ENF_CNAND` returns true if the result is false; `ENF_CNOR` returns false if the result is true. Unlike the other binary operators, these four do not leave a result on the stack, even if they continue.

The short-circuit operators should be used when possible, to reduce the amount of time spent evaluating filters. When they are used, you should also arrange the order of the tests so that the filter will succeed or fail as soon as possible; for example, checking the IP destination field of a UDP packet is more likely to indicate failure than the packet type field.

The special action `ENF_NOPUSH` and the special operator `ENF_NOP` can be used to only perform the binary operation or to only push a value on the stack. Since both are (conveniently)

defined to be zero, indicating only an action actually specifies the action followed by ENF_NOP, and indicating only an operation actually specifies ENF_NOPUSH followed by the operation.

After executing the filter command list, a non-zero value (true) left on top of the stack (or an empty stack) causes the incoming packet to be accepted and a zero value (false) causes the packet to be rejected. (If the filter exits as the result of a short-circuit operator, the top-of-stack value is ignored.) Specifying an undefined operation or action in the command list or performing an illegal operation or action (such as pushing a shortword offset past the end of the packet or executing a binary operator with fewer than two shortwords on the stack) causes a filter to reject the packet.

Examples The packet filter module is not dependent on any particular device driver or module but is commonly used with datalink drivers such as the Ethernet driver. If the underlying datalink driver supports the Data Link Provider Interface (DLPI) message set, the appropriate STREAMS DLPI messages must be issued to attach the stream to a particular hardware device and bind a datalink address to the stream before the underlying driver will route received packets upstream. Refer to the DLPI Version 2 specification for details on this interface.

The reverse ARP daemon program may use code similar to the following fragment to construct a filter that rejects all but RARP packets. That is, it accepts only packets whose Ethernet type field has the value ETHERTYPE_REVARP. The filter works whether a VLAN is configured or not.

```
struct ether_header eh;        /* used only for offset values */
struct packetfilt pf;
register ushort_t *fwp = pf.Pf_Filter;
ushort_t offset;
int    fd;
/*
 * Push packet filter streams module.
 */
if (ioctl(fd, I_PUSH, "pfmod") < 0)
    syserr("pfmod");

/*
 * Set up filter.  Offset is the displacement of the Ethernet
 * type field from the beginning of the packet in units of
 * ushort_ts.
 */
offset = ((uint_t) &eh.ether_type - (uint_t) &eh.ether_dhost) /
                sizeof (us_short);
       *fwp++ = ENF_PUSHWORD + offset;
       *fwp++ = ENF_PUSHLIT;
       *fwp++ = htons(ETHERTYPE_VLAN);
       *fwp++ = ENF_EQ;
       *fwp++ = ENF_BRFL;
       *fwp++ = 3;  /* If this isn't ethertype VLAN, don't change oreg */
```

```
*fwp++ = ENF_LOAD_OFFSET;
*fwp++ = 2;  /* size of the VLAN tag in words */
*fwp++ = ENF_POP;
*fwp++ = ENF_PUSHWORD + offset;
*fwp++ = ENF_PUSHLIT;
*fwp++ = htons(ETHERTYPE_REVARP);
*fwp++ = ENF_EQ;
pf.Pf_FilterLen = fwp - &pf.PF_Filter[0];
```

This filter can be abbreviated by taking advantage of the ability to combine actions and operations:

```
*fwp++ = ENF_PUSHWORD + offset;
*fwp++ = ENF_PUSHLIT | ENF_EQ;
*fwp++ = htons(ETHERTYPE_REVARP);
*fwp++ = htons(ETHERTYPE_VLAN);
*fwp++ = ENF_BRFL | ENF_NOP;
*fwp++ = 3;
*fwp++ = ENF_LOAD_OFFSET | ENF_NOP;
*fwp++ = 2;
*fwp++ = ENF_POP | ENF_NOP;
*fwp++ = ENF_PUSHWORD + offset;
*fwp++ = ENF_PUSHLIT | ENF_EQ;
*fwp++ = htons(ETHERTYPE_REVARP);
```

See Also bufmod(7M), dlpi(7P), pullupmsg(9F)

Name physmem – physmem driver

Description The `physmem` driver is a private mechanism used by diagnostic test suites to test the physical memory of the system.

Files `/dev/physmem`
Kernel module.

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface stability	Private

Caution This driver is used by Sun internal diagnostic programs only. Any other use may have a harmful impact on the system.

Name pipemod – STREAMS pipe flushing module

Description The typical stream is composed of a stream head connected to modules and terminated by a driver. Some stream configurations such as pipes and FIFOs do not have a driver and hence certain features commonly supported by the driver need to be provided by other means. Flushing is one such feature, and it is provided by the pipemod module.

Pipes and FIFOs in their simplest configurations only have stream heads. A write side is connected to a read side. This remains true when modules are pushed. The twist occurs at a point known as the mid-point. When an M_FLUSH message is passed from a write queue to a read queue the FLUSHR and/or FLUSHW bits have to be switched. The mid-point of a pipe is not always easily detectable, especially if there are numerous modules pushed on either end of the pipe. In that case there needs to be a mechanism to intercept all message passing through the stream. If the message is an M_FLUSH message and it is at the mid-point, the flush bits need to be switched. This bit switching is handled by the pipemod module.

pipemod should be pushed onto a pipe or FIFO where flushing of any kind will take place. The pipemod module can be pushed on either end of the pipe. The only requirement is that it is pushed onto an end that previously did not have modules on it. That is, pipemod must be the first module pushed onto a pipe so that it is at the mid-point of the pipe itself.

The pipemod module handles only M_FLUSH messages. All other messages are passed on to the next module using the putnext() utility routine. If an M_FLUSH message is passed to pipemod and the FLUSHR and FLUSHW bits are set, the message is not processed but is passed to the next module using the putnext() routine. If only the FLUSHR bit is set, the FLUSHR bit is turned off and the FLUSHW bit is set. The message is then passed on to the next module using putnext(). Similarly, if the FLUSHW bit is the only bit set in the M_FLUSH message, the FLUSHW bit is turned off and the FLUSHR bit is turned on. The message is then passed to the next module on the stream.

The pipemod module can be pushed on any stream that desires the bit switching. It must be pushed onto a pipe or FIFO if any form of flushing must take place.

See Also *STREAMS Programming Guide*

Name pm – Power Management driver

Synopsis `/dev/pm`

Description The Power Management (`pm`) driver provides an interface for applications to configure devices within the system for Power Management. The interface is provided through `ioctl(2)` commands. The `pm` driver may be accessed using `/dev/pm`.

Power Management Framework

The Power Management framework model allows the system to be viewed as a collection of devices. Each device is a collection of components that comprise the smallest power manageable units. The device driver controls the definition of a device's power manageable components.

A component can either be *busy* or *idle* at the current power level. Normally, the Power Management framework takes an *idle* component to the next lower power level. The Power Management framework uses two factors to determine this transition: the component must have been idle for at least the threshold time, and the device to which the component belongs must satisfy any dependency requirements. A dependency occurs when a device requires another device to be power managed before it can be power managed. Dependencies occur on a per device basis: when a dependency exists, no components of a device may be managed unless all the devices it depends upon are first power managed.

Using the commands below, an application may take control of the Power Management of a device from the Power Management framework driver and manage the transition of device power levels directly.

For this set of ioctl commands, *arg* (see `ioctl(2)`) points to a structure of type `pm_req` defined in `<sys/pm.h>`:

```
typedef struct pm_req {
        char *physpath;     /* physical path of device  */
                            /* to configure. See libdevinfo(3LIB) */
        int  component;     /* device component   */
        int  value;         /* power level, threshold value, or count */
        void *data;         /* command-dependent variable-sized data */
        size_t  datasize;   /* size of data buffer */
    } pm_req_t;
```

The fields should contain the following data:

physpath Pointer to the physical path of a device. See `libdevinfo(3LIB)`. For example, for the device `/devices/pseudo/pm@0:pm` the *physpath* value would be `/pseudo/pm@0`.

component Non-negative integer specifying which component is being configured. The numbering starts at zero.

value Non-negative integer specifying the threshold value in seconds or the desired power level, or the number of levels being specified.

data Pointer to a buffer which contains or receives variable-sized data, such as the name of a device upon which this device has a dependency.

size Size of the data buffer.

Not all fields are used in each command.

`PM_DIRECT_PM` The device named by *physpath* is disabled from being power managed by the framework. The caller will power manage the device directly using the `PM_DIRECT_NOTIFY`, `PM_GET_TIME_IDLE` and `PM_GET_CURRENT_POWER`, `PM_GET_FULL_POWER` and `PM_SET_CURRENT_POWER` commands. If the device needs to have its power level changed either because its driver calls `pm_raise_power(9F)`, `pm_lower_power(9F)`, or `pm_power_has_changed(9F)` or because the device is the parent of another device that is changing power level or a device that this device depends on is changing power level, then the power level change of the device will be blocked and the caller will be notified as described below for the `PM_DIRECT_NOTIFY` command.

Error codes:

`EBUSY` Device already disabled for Power Management by framework.

`EPERM` Caller is neither superuser nor effective group ID of 0.

`PM_RELEASE_DIRECT_PM` The device named by *physpath* (which must have been the target of a `PM_DIRECT_PM` command) is re-enabled for Power Management by the framework.

Error codes:

`EINVAL` Device component out of range.

`PM_DIRECT_NOTIFY PM_DIRECT_NOTIFY_WAIT`

These commands allow the process that is directly power managing a device to be notified of events that could change the power level of the device. When such an event occurs, this command returns information about the event.

arg (see `ioctl(2)`) points to a structure of type `pm_state_change` defined in `<sys/pm.h>`:

```
typedef struct  pm_state_change {
   char   *physpath;    /* device which has changed state */
   int     component;   /* which component changed state */
#if defined(_BIG_ENDIAN)
    ushort_t flags;      /* PSC_EVENT_LOST, PSC_ALL_LOWEST */
    ushort_t event;      /* type of event */
#else
    ushort_t event;      /* type of event *
    ushort_t flags;      /* PSC_EVENT_LOST, PSC_ALL_LOWEST */
#endif
    time_t  timestamp;   /* time of state change */+
    int     old_level;    /* power level changing from */
    int    new_level;   /* power level changing to */
    size_t  size;        /* size of buffer physpath points to */
} pm_state_change_t;
```

When an event occurs, the struct pointed to by *arg* is filled in. If the event type is `PSC_PENDING_CHANGE`, then the information in the rest of the struct describes an action that the framework would have taken if the device were not directly power managed by the caller. The caller is responsible for completing the indicated level changes using `PM_SET_CURRENT_POWER` below.

An event type of `PSC_HAS_CHANGED` indicates that the driver for the directly power managed device has called `pm_power_has_changed(9F)` due to the device changing power on its own. It is provided to allow the caller to track the power state of the device.

The system keeps events in a circular buffer. If the buffer overflow, the oldest events are lost and when the event that next follows a lost event is retrieved it will have PSC_EVENT_LOST set in flags.

`PM_DIRECT_NOTIFY` returns `EWOULDBLOCK` if no event is pending, and `PM_DIRECT_NOTIFY_WAIT` blocks until an event is available.

`pm` also supports the `poll(2)` interface. When an event is pending a `poll(2)` call that includes a file descriptor for `/dev/pm` and that has `POLLIN` or `POLLRDNORM` set in its event mask will return.

`PM_SET_CURRENT_POWER`

Component *component* of the device named by *physpath* (which must contain the physical path of a device against which the process has issued a `PM_DIRECT_PM` command) is set to power level *value.* If all components of the device named by *physpath* were at level 0, *value* is non-zero and some device has a dependency on this device, then all components of that device will be brought to full power before this command returns. Similarly, if the parent of the target device is powered off, then it will be brought up as needed before this command returns. When PM_SET_CURRENT_POWER is issued against a device, the resulting power change is included in the event list for PM_DIRECT_NOTIFY.

Error codes:

`EINVAL` Device component out of range, or power level < 0.

`EIO` Failed to power device or its ancestors or the devices on which this device has dependency or their ancestors. Note that this may not indicate

a failure, the device driver may have rejected the command as inappropriate because the component has become busy.

`EPERM` Caller has not previously issued a successful `PM_DIRECT_PM` command against this device.

`PM_GET_FULL_POWER` The highest supported power level of component *component* of the device named by *physpath* is returned.

`PM_GET_CURRENT_POWER` The current power level of component *component* of the device named by *physpath* is returned.

Error codes:

`EAGAIN` Device component power level is not currently known.

`PM_GET_TIME_IDLE` `PM_GET_TIME_IDLE` returns the number of seconds that component *component* of the device named by *physpath* has been idle. If the device is not idle, then `0` is returned.

Note that because the state of the device may change between the time the process issues the `PM_GET_TIME_IDLE` command and the time the process issues a `PM_SET_CURRENT_POWER` command to reduce the power level of an idle component, the process must be prepared to deal with a `PM_SET_CURRENT_POWER` command returning failure because the driver has rejected the command as inappropriate because the device component has become busy. This can be differentiated from other types of failures by issuing the `PM_GET_TIME_IDLE` command again to see if the component has become busy.

Errors Upon error, the commands will return −1, and set *errno*. In addition to the error codes listed above by command, the following error codes are common to all commands:

EFAULT	Bad address passed in as argument.
ENODEV	Device is not power manageable, or device is not configured.
ENXIO	Too many opens attempted.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface stability	Unstable

See Also pmconfig(1M), Intro(2), ioctl(2), libdevinfo(3LIB), power.conf(4), attributes(5), attach(9E), detach(9E), power(9E), pm_busy_component(9F), pm_idle_component(9F), pm_lower_power(9F), pm_power_has_changed(9F), pm_raise_power(9F)

Writing Device Drivers

Name poll – driver for fast poll on many file descriptors

Synopsis

```
#include <sys/devpoll.h>
int fd = open("/dev/poll", O_RDWR);
ssize_t n = write(int fd, struct pollfd buf[], int bufsize);
int n = ioctl(int fd, DP_POLL, struct dvpoll* arg);
int n = ioctl(int fd, DP_ISPOLLED, struct pollfd* pfd);
```

Parameters

fd	Open file descriptor that refers to the `/dev/poll` driver.
path	`/dev/poll`
buf	Array of `pollfd` structures.
bufsize	Size of *buf* in bytes.
arg	Pointer to `pollcall` structure.
pfd	Pointer to `pollfd` structure.

Description **Note –** The `/dev/poll` device, associated driver and corresponding manpages may be removed in a future Solaris release. For similar functionality in the event ports framework, see `port_create(3C)`.

The `/dev/poll` driver is a special driver that enables you to monitor multiple sets of polled file descriptors. By using the `/dev/poll` driver, you can efficiently poll large numbers of file descriptors. Access to the `/dev/poll` driver is provided through `open(2)`, `write(2)`, and `ioctl(2)` system calls.

Writing an array of `pollfd` struct to the `/dev/poll` driver has the effect of adding these file descriptors to the monitored `poll` file descriptor set represented by the *fd*. To monitor multiple file descriptor sets, open the `/dev/poll` driver multiple times. Each `fd` corresponds to one set. For each `pollfd` struct entry (defined in `sys/poll.h`):

```
struct pollfd {
   int  fd;
   short events;
   short revents;
}
```

The `fd` field specifies the file descriptor being polled. The `events` field indicates the interested `poll` `events` on the file descriptor. If a `pollfd` array contains multiple `pollfd` entries with the same `fd` field, the "events" field in each `pollfd` entry is OR'ed. A special `POLLREMOVE` event in the `events` field of the `pollfd` structure removes the `fd` from the monitored set. The `revents` field is not used. Write returns the number of bytes written successfully or `-1` when write fails.

The `DP_POLL` ioctl is used to retrieve returned `poll` `events` occured on the polled file descriptors in the monitored set represented by *fd*. *arg is a* pointer to the devpoll structures which are defined as follows:

```
struct dvpoll {
    struct pollfd* dp_fds;
    int dp_nfds;
    int dp_timeout;
}
```

The `dp_fds` points to a buffer that holds an array of returned `pollfd` structures. The `dp_nfds` field specifies the size of the buffer in terms of the number of `pollfd` entries it contains. The `dp_nfds` field also indicates the maximum number of file descriptors from which poll information can be obtained. If there is no interested `events` on any of the polled file descriptors, the `DP_POLL` ioctl call will wait `dp_timeout` milliseconds before returning. If `dp_timeout` is `0`, the ioctl call returns immediately. If `dp_timeout` is `-1`, the call blocks until an interested `poll events` is available or the call is interrupted. Upon return, if the ioctl call has failed, `-1` is returned. The memory content pointed by `dp_fds` is not modified. A return value `0` means the ioctl is timed out. In this case, the memory content pointed by `dp_fds` is not modified. If the call is successful, it returns the number of valid `pollfd` entries in the array pointed by `dp_fds`; the contents of the rest of the buffer is undefined. For each valid `pollfd` entry, the `fd` field indicates the file desciptor on which the polled `events` happened. The `events` field is the user specified `poll events`. The `revents` field contains the `events` occurred. –`1` is returned if the call fails.

`DP_ISPOLLED` ioctl allows you to query if a file descriptor is already in the monitored set represented by `fd`. The `fd` field of the `pollfd` structure indicates the file descriptor of interest. The `DP_ISPOLLED` ioctl returns `1` if the file descriptor is in the set. The `events` field contains `0`. The `revents` field contains the currently polled `events`. The ioctl returns `0` if the file descriptor is not in the set. The `pollfd` structure pointed by *pfd* is not modified. The ioctl returns a `-1` if the call fails.

Examples The following example shows how `/dev/poll` may be used.

```
{
        ...
        /*
         * open the driver
         */
        if ((wfd = open("/dev/poll", O_RDWR)) < 0) {
                exit(-1);
        }
        pollfd = (struct pollfd* )malloc(sizeof(struct pollfd) * MAXBUF);
        if (pollfd == NULL) {
                close(wfd);
                exit(-1);
        }
        /*
         * initialize buffer
         */
        for (i = 0; i < MAXBUF; i++) {
```

```
                pollfd[i].fd = fds[i];
                pollfd[i].events = POLLIN;
                pollfd[i].revents = 0;
        }
        if (write(wfd, &pollfd[0], sizeof(struct pollfd) * MAXBUF) !=
                        sizeof(struct pollfd) * MAXBUF) {
                perror("failed to write all pollfds");
                close (wfd);
                free(pollfd);
                exit(-1);
        }
        /*
         * read from the devpoll driver
         */
        dopoll.dp_timeout = -1;
        dopoll.dp_nfds = MAXBUF;
        dopoll.dp_fds = pollfd;
        result = ioctl(wfd, DP_POLL, &dopoll);
        if (result < 0) {
                perror("/dev/poll ioctl DP_POLL failed");
                close (wfd);
                free(pollfd);
                exit(-1);
        }
        for (i = 0; i < result; i++) {
                read(dopoll.dp_fds[i].fd, rbuf, STRLEN);
        }
 ...
}
```

The following example is part of a test program which shows how `DP_ISPOLLED()` ioctl may be used.

```
{
    ...

        loopcnt = 0;
        while (loopcnt < ITERATION) {
                rn = random();
                rn %= RANGE;
                if (write(fds[rn], TESTSTRING, strlen(TESTSTRING)) !=
                                strlen(TESTSTRING)) {
                        perror("write to fifo failed.");
                        close (wfd);
                        free(pollfd);
                        error = 1;
                        goto out1;
                }
```

```
                dpfd.fd = fds[rn];
                dpfd.events = 0;
                dpfd.revents = 0;
                result = ioctl(wfd, DP_ISPOLLED, &dpfd);
                if (result < 0) {
                        perror("/dev/poll ioctl DP_ISPOLLED failed");
                        printf("errno = %d\n", errno);
                        close (wfd);
                        free(pollfd);
                        error = 1;
                        goto out1;
                }
                if (result != 1) {
                        printf("DP_ISPOLLED returned incorrect result: %d.\n",
                                result);
                        close (wfd);
                        free(pollfd);
                        error = 1;
                        goto out1;
                }
                if (dpfd.fd != fds[rn]) {
                        printf("DP_ISPOLLED returned wrong fd %d, expect %d\n",
                                dpfd.fd, fds[rn]);
                        close (wfd);
                        free(pollfd);
                        error = 1;
                        goto out1;
        }
                if (dpfd.revents != POLLIN) {
                        printf("DP_ISPOLLED returned unexpected revents %d\n",
                                dpfd.revents);
                        close (wfd);
                        free(pollfd);
                        error = 1;
                        goto out1;
                }
                if (read(dpfd.fd, rbuf, strlen(TESTSTRING)) !=
                                strlen(TESTSTRING)) {
                        perror("read from fifo failed");
                        close (wfd);
                        free(pollfd);
                        error = 1;
                        goto out1;
                }
                loopcnt++;
        }
```

Errors

EACCES	A process does not have permission to access the content cached in /dev/poll.
EINTR	A signal was caught during the execution of the ioctl(2) function.
EFAULT	The request argument requires a data transfer to or from a buffer pointed to by *arg*, but *arg* points to an illegal address.
EINVAL	The request or *arg* parameter is not valid for this device, or field of the dvpoll struct pointed by *arg* is not valid (for example, when using write/pwrite dp_nfds is greater than {OPEN_MAX}, or when using the DPPOLL ioctl dp_nfds is greater than or equal to {OPEN_MAX}}.
ENXIO	The O_NONBLOCK flag is set, the named file is a FIFO, the O_WRONLY flag is set, and no process has the file open for reading; or the named file is a character special or block special file and the device associated with this special file does not exist.

Attributes See attributes(5) for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86
Availability	SUNWcar (Solaris)
	SUNWcsr, SUNWcsu (Solaris on x86)
	SUNWhea (header files)
Interface Stability	Obsolete
MT-Level	Safe

See Also open(2), poll(2), write(2), attributes(5)

Notes The /dev/poll API is particularly beneficial to applications that poll a large number of file descriptors repeatedly. Applications will exhibit the best performance gain if the polled file descriptor list rarely change.

When using the /dev/poll driver, you should remove a closed file descriptor from a monitored poll set. Failure to do so may result in a POLLNVAL revents being returned for the closed file descriptor. When a file descriptor is closed but not removed from the monitored set, and is reused in subsequent open of a different device, you will be polling the device associated with the reused file descriptor. In a multithreaded application, careful coordination among threads doing close and DP_POLL ioctl is recommended for consistent results.

The /dev/poll driver caches a list of polled file descriptors, which are specific to a process. Therefore, the /dev/poll file descriptor of a process will be inherited by its child process, just like any other file descriptors. But the child process will have very limited access through this

inherited `/dev/poll` file descriptor. Any attempt to write or do ioctl by the child process will result in an `EACCES` error. The child process should close the inherited `/dev/poll` file descriptor and open its own if desired.

The `/dev/poll` driver does not yet support polling. Polling on a `/dev/poll` file descriptor will result in `POLLERR` being returned in the `revents` field of `pollfd` structure.

Name prnio – generic printer interface

Synopsis

```
#include <sys/prnio.h>
```

Description The `prnio` generic printer interface defines ioctl commands and data structures for printer device drivers.

`prnio` defines and provides facilities for five basic phases of the printing process:

- Identification — Retrieve device information/attributes
- Setup — Set device attributes
- Transfer — Transfer data to or from the device
- Cleanup — Transfer phase conclusion
- Abort — Transfer phase interruption

During the Identification phase, the application retrieves a set of device capabilities and additional information using the `PRNIOC_GET_IFCAP`, `PRNIOC_GET_STATUS`, `PRNIOC_GET_TIMEOUTS`, `PRNIOC_GET_IFINFO` and `PRNIOC_GET_1284_DEVID` commands.

During the Setup phase the application sets some interface attributes and probably resets the printer as described in the `PRNIOC_SET_IFCAP`, `PRNIOC_SET_TIMEOUTS` and `PRNIOC_RESET` sections.

During the Transfer phase, data is transferred in a forward (host to peripheral) or reverse direction (peripheral to host). Transfer is accomplished using `write(2)` and `read(2)` system calls. For `prnio` compliant printer drivers, forward transfer support is mandatory, while reverse transfer support is optional. Applications can also use `PRNIOC_GET_STATUS` and `PRNIOC_GET_1284_STATUS` commands during the transfer to monitor the device state.

The Cleanup phase is accomplished by closing the device using `close(2)`. Device drivers supporting `prnio` may set non-zero error code as appropriate. Applications should explicitly `close(2)` a device before exiting and check `errno` value.

The Abort phase is accomplished by interrupting the `write(2)` and `read(2)` system calls. The application can perform some additional cleanup during the Abort phase as described in `PRNIOC_GET_IFCAP` section.

ioctls

`PRNIOC_GET_IFCAP` Application can retrieve printer interface capabilities using this command. The `ioctl(2)` argument is a pointer to `uint_t`, a bit field representing a set of properties and services provided by a printer driver. Set bit means supported capability. The following values are defined:

`PRN_BIDI` - When this bit is set, the interface operates in a bidirectional mode, instead of forward-only mode.
`PRN_HOTPLUG` - If this bit is set, the interface allows device hot-plugging.

PRN_1284_DEVID - If this bit is set, the device is capable of returning *1284* device ID (see PRNIOC_GET_1284_DEVID.)
PRN_1284_STATUS - If this bit is set, the device driver can return device status lines (see PRNIOC_GET_1284_STATUS). Some devices support this ioctl in unidirectional mode only.
PRN_TIMEOUTS - If this bit is set the peripheral may stall during the transfer phase and the driver can timeout and return from the write(2) and read(2) returning the number of bytes that have been transferred. If PRN_TIMEOUTS is set, the driver supports this functionality and the timeout values can be retrieved and modified via the PRNIOC_GET_TIMEOUTS and PRNIOC_SET_TIMEOUTS ioctls. Otherwise, applications can implement their own timeouts and abort phase.
PRN_STREAMS - This bit impacts the application abort phase behaviour. If the device claimed PRN_STREAMS capability, the application must issue an I_FLUSH ioctl(2) before close(2) to dismiss the untransferred data. Only STREAMS drivers can support this capability.

PRNIOC_SET_IFCAP — This ioctl can be used to change interface capabilities. The argument is a pointer to uint_t bit field that is described in detail in the PRNIOC_GET_IFCAP section. Capabilities should be set one at a time; otherwise the command will return EINVAL. The following capabilities can be changed by this ioctl:

PRN_BIDI - When this capability is set, the interface operates in a bidirectional mode, instead of forward-only mode. Devices that support only one mode will not return error; applications should use PRNIOC_GET_IFCAP to check if the mode was successfully changed. Because some capabilities may be altered as a side effect of changing other capabilities, this command should be followed by PRNIOC_GET_IFCAP.

PRNIOC_GET_IFINFO — This command can be used to retrieve printer interface info string, which is an arbitrary format string usually describing the bus type. The argument is a pointer to struct prn_interface_info as described below.

```
struct prn_interface_info {
  uint_t      if_len;   /* length of buffer */
  uint_t      if_rlen;  /* actual info length */
  char   *if_data;  /* buffer address */
};
```

The application allocates a buffer and sets `if_data` and `if_len` values to its address and length, respectively. The driver returns the string to this buffer and sets `if_len` to its length. If `if_len` is less that `if_rlen`, the driver must return the first `if_len` bytes of the string. The application may then repeat the command with a bigger buffer.

Although `prnio` does not limit the contents of the interface info string, some values are recommended and defined in <`sys/prnio.h`> by the following macros:

`PRN_PARALLEL` - Centronics or *IEEE 1284* compatible devices
`PRN_SERIAL` - EIA-232/EIA-485 serial ports
`PRN_USB` - Universal Serial Bus printers
`PRN_1394` - *IEEE 1394* peripherals
Printer interface info string is for information only: no implications should be made from its value.

PRNIOC_RESET

Some applications may want to reset the printer state during Setup and/or Cleanup phase using `PRNIOC_RESET` command. Reset semantics are device-specific, and in general, applications using this command should be aware of the printer type.

Each `prnio` compliant driver is required to accept this request, although performed actions are completely driver-dependent. More information on the `PRNIOC_RESET` implementation for the particular driver is available in the corresponding man page and printer manual.

PRNIOC_GET_1284_DEVID

This command can be used to retrieve printer device ID as defined by *IEEE 1284-1994*.The `ioctl(2)` argument is a pointer to `struct prn_1284_device_id` as described below.

```
struct prn_1284_device_id {
   uint_t      id_len;   /* length of buffer */
   uint_t      id_rlen;  /* actual ID length */
   char      *id_data;  /* buffer address */
};
```

For convenience, the two-byte length field is not considered part of device ID string and is not returned in the user buffer. Instead, `id_rlen` value shall be set to (length - 2) by the driver, where length is the ID length field value. If buffer length is less than `id_rlen`, the driver returns the first `id_len` bytes of the ID.

The printer driver must return the most up-to-date value of the device ID.

PRNIOC_GET_STATUS — This command can be used by applications to retrieve current device status. The argument is a pointer to `uint_t`, where the status word is returned. Status is a combination of the following bits:

`PRN_ONLINE` - For devices that support `PRN_HOTPLUG` capability, this bit is set when the device is online, otherwise the device is offline. Devices without `PRN_HOTPLUG` support should always have this bit set.
`PRN_READY` - This bit indicates if the device is ready to receive/send data. Applications may use this bit for an outbound flow control

`PRNIOC_GET_1284_STATUS` — Devices that support `PRN_1284_STATUS` capability accept this ioctl to retrieve the device status lines defined in *IEEE 1284* for use in Compatibility mode. The following bits may be set by the driver:

`PRN_1284_NOFAULT` - Device is not in error state
`PRN_1284_SELECT` - Device is selected
`PRN_1284_PE` - Paper error
`PRN_1284_BUSY` - Device is busy

`PRNIOC_GET_TIMEOUTS` — This command retrieves current transfer timeout values for the driver. The argument is a pointer to `struct prn_timeouts` as described below.

```
struct prn_timeouts {
  uint_t    tmo_forward;  /* forward transfer timeout */
  uint_t    tmo_reverse;  /* reverse transfer timeout */
    };
```

`tmo_forward` and `tmo_reverse` define forward and reverse transfer timeouts in seconds. This command is only valid for drivers that support `PRN_TIMEOUTS` capability.

`PRNIOC_SET_TIMEOUTS` — This command sets current transfer timeout values for the driver. The argument is a pointer to `struct prn_timeouts`. See `PRNIOC_GET_TIMEOUTS` for description of this structure. This command is only valid for drivers that support `PRN_TIMEOUTS` capability.

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, IA

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

See Also close(2), ioctl(2), read(2), write(2), attributes(5), ecpp(7D), usbprn(7D), lp(7D)

IEEE Std 1284-1994

Name profile – DTrace profile interrupt provider

Description The `profile` driver is a DTrace dynamic tracing provider that adds time-based interrupt event sources that can be used as DTrace probes.

Each profile event source is a time-based interrupt firing every fixed, specified time interval. You can use these probes to sample some aspect of system state every unit time and the samples can then be used to infer system behavior. If the sampling rate is high, or the sampling time is long, an accurate inference is possible. By using the DTrace facility to bind arbitrary actions to probes, you can use the `profile` provider to sample practically anything in the system. For example, you could sample the state of the current thread, the CPU state, or the current machine instruction each time a probe fires.

The `profile` driver is not a public interface and you access the instrumentation offered by this provider through DTrace. Refer to the *Solaris Dynamic Tracing Guide* for a description of the public documented interfaces available for the DTrace facility and the probes offered by the profile provider.

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWdtrp
Interface Stability	Private

See Also `dtrace(1M)`, `attributes(5)`, `dtrace(7D)`

Solaris Dynamic Tracing Guide

Name ptem – STREAMS Pseudo Terminal Emulation module

Synopsis

```
int ioctl(fd, I_PUSH, "ptem");
```

Description `ptem` is a STREAMS module that, when used in conjunction with a line discipline and pseudo terminal driver, emulates a terminal.

The `ptem` module must be pushed (see `I_PUSH`, `streamio(7I)`) onto the slave side of a pseudo terminal STREAM, before the `ldterm(7M)` module is pushed.

On the write-side, the `TCSETA`, `TCSETAF`, `TCSETAW`, `TCGETA`, `TCSETS`, `TCSETSW`, `TCSETSF`, `TCGETS`, `TCSBRK`, `JWINSIZE`, `TIOCGWINSZ`, and `TIOCSWINSZ` `termio` `ioctl(2)` messages are processed and acknowledged. If remote mode is not in effect, `ptem` handles the `TIOCSTI` ioctl by copying the argument bytes into an `M_DATA` message and passing it back up the read side. Regardless of the remote mode setting, `ptem` acknowledges the ioctl and passes a copy of it downstream for possible further processing. A hang up (that is, `stty 0`) is converted to a zero length `M_DATA` message and passed downstream. Termio `cflags` and window row and column information are stored locally one per stream. `M_DELAY` messages are discarded. All other messages are passed downstream unmodified.

On the read-side all messages are passed upstream unmodified with the following exceptions. All `M_READ` and `M_DELAY` messages are freed in both directions. A `TCSBRK` ioctl is converted to an `M_BREAK` message and passed upstream and an acknowledgement is returned downstream. A `TIOCSIGNAL` ioctl is converted into an `M_PCSIG` message, and passed upstream and an acknowledgement is returned downstream. Finally a `TIOCREMOTE` ioctl is converted into an `M_CTL` message, acknowledged, and passed upstream; the resulting mode is retained for use in subsequent `TIOCSTI` parsing.

Files `<sys/ptem.h>`

See Also `stty(1)`, `ioctl(2)`, `ldterm(7M)`, `pckt(7M)`, `streamio(7I)`, `termio(7I)`

STREAMS Programming Guide

Name ptm – STREAMS pseudo-tty master driver

Description The pseudo-tty subsystem simulates a terminal connection, where the master side represents the terminal and the slave represents the user process's special device end point. In order to use the pseudo-tty subsystem, a node for the master side driver `/dev/ptmx` and `N` number of nodes for the slave driver must be installed. See `pts(7D)`. The master device is set up as a cloned device where its major device number is the major for the clone device and its minor device number is the major for the `ptm` driver. There are no nodes in the file system for master devices. The master pseudo driver is opened using the `open(2)` system call with `/dev/ptmx` as the device parameter. The clone open finds the next available minor device for the `ptm` major device.

A master device is available only if it and its corresponding slave device are not already open. When the master device is opened, the corresponding slave device is automatically locked out. Only one open is allowed on a master device. Multiple opens are allowed on the slave device. After both the master and slave have been opened, the user has two file descriptors which are the end points of a full duplex connection composed of two streams which are automatically connected at the master and slave drivers. The user may then push modules onto either side of the stream pair.

The master and slave drivers pass all messages to their adjacent queues. Only the `M_FLUSH` needs some processing. Because the read queue of one side is connected to the write queue of the other, the `FLUSHR` flag is changed to the `FLUSHW` flag and vice versa. When the master device is closed an `M_HANGUP` message is sent to the slave device which will render the device unusable. The process on the slave side gets the errno `EIO` when attempting to write on that stream but it will be able to read any data remaining on the stream head read queue. When all the data has been read, `read()` returns 0 indicating that the stream can no longer be used. On the last close of the slave device, a 0-length message is sent to the master device. When the application on the master side issues a `read()` or `getmsg()` and 0 is returned, the user of the master device decides whether to issue a `close()` that dismantles the pseudo-terminal subsystem. If the master device is not closed, the pseudo-tty subsystem will be available to another user to open the slave device.

If `O_NONBLOCK` or `O_NDELAY` is set, read on the master side returns –1 with errno set to `EAGAIN` if no data is available, and write returns –1 with errno set to `EAGAIN` if there is internal flow control.

ioctls The master driver supports the `ISPTM` and `UNLKPT` ioctls that are used by the functions `grantpt(3C)`, `unlockpt(3C)` and `ptsname(3C)`. The ioctl `ISPTM` determines whether the file descriptor is that of an open master device. On success, it returns the 0. The ioctl `UNLKPT` unlocks the master and slave devices. It returns 0 on success. On failure, the errno is set to `EINVAL` indicating that the master device is not open.

Files

`/dev/ptmx`	master clone device
`/dev/pts/M`	slave devices (M = 0 -> N-1)

See Also `grantpt`(3C), `ptsname`(3C), `unlockpt`(3C), `pckt`(7M), `pts`(7D)

STREAMS Programming Guide

Name pts – STREAMS pseudo-tty slave driver

Description The pseudo-tty subsystem simulates a terminal connection, where the master side represents the terminal and the slave represents the user process's special device end point. In order to use the pseudo-tty subsystem, a node for the master side driver `/dev/ptmx` and N nodes for the slave driver (N is determined at installation time) must be installed. The names of the slave devices are `/dev/pts/M` where `M` has the values 0 through N-1. When the master device is opened, the corresponding slave device is automatically locked out. No user may open that slave device until its permissions are adjusted and the device unlocked by calling functions `grantpt(3C)` and `unlockpt(3C)`. The user can then invoke the open system call with the name that is returned by the `ptsname(3C)` function. See the example below.

Only one open is allowed on a master device. Multiple opens are allowed on the slave device. After both the master and slave have been opened, the user has two file descriptors which are end points of a full duplex connection composed of two streams automatically connected at the master and slave drivers. The user may then push modules onto either side of the stream pair. The user needs to push the `ptem(7M)` and `ldterm(7M)` modules onto the slave side of the pseudo-terminal subsystem to get terminal semantics.

The master and slave drivers pass all messages to their adjacent queues. Only the `M_FLUSH` needs some processing. Because the read queue of one side is connected to the write queue of the other, the `FLUSHR` flag is changed to the `FLUSHW` flag and vice versa. When the master device is closed an `M_HANGUP` message is sent to the slave device which will render the device unusable. The process on the slave side gets the errno `EIO` when attempting to write on that stream but it will be able to read any data remaining on the stream head read queue. When all the data has been read, read returns 0 indicating that the stream can no longer be used. On the last close of the slave device, a 0-length message is sent to the master device. When the application on the master side issues a `read()` or `getmsg()` and 0 is returned, the user of the master device decides whether to issue a `close()` that dismantles the pseudo-terminal subsystem. If the master device is not closed, the pseudo-tty subsystem will be available to another user to open the slave device. Since 0-length messages are used to indicate that the process on the slave side has closed and should be interpreted that way by the process on the master side, applications on the slave side should not write 0-length messages. If that occurs, the write returns 0, and the 0-length message is discarded by the `ptem` module.

The standard STREAMS system calls can access the pseudo-tty devices. The slave devices support the `O_NDELAY` and `O_NONBLOCK` flags.

Examples

```
int    fdm fds;
char   *slavename;
extern char *ptsname();

fdm = open("/dev/ptmx", O_RDWR);  /* open master */
grantpt(fdm);                     /* change permission of    slave */
unlockpt(fdm);                    /* unlock slave */
slavename = ptsname(fdm);         /* get name of slave */
```

```
fds = open(slavename, O_RDWR);    /* open slave */
ioctl(fds, I_PUSH, "ptem");       /* push ptem */
ioctl(fds, I_PUSH, "ldterm");     /* push ldterm*/
```

Files

`/dev/ptmx` master clone device

`/dev/pts/M` slave devices (M = 0 -> N-1)

See Also `grantpt`(3C), `ptsname`(3C), `unlockpt`(3C), `ldterm`(7M), `ptm`(7D), `ptem`(7M)

STREAMS Programming Guide

Name pty – pseudo-terminal driver

Description The `pty` driver provides support for a pair of devices collectively known as a *pseudo-terminal*. The two devices comprising a pseudo-terminal are known as a *controller* and a *slave*. The slave device distinguishes between the `B0` baud rate and other baud rates specified in the `c_cflag` word of the `termios` structure, and the `CLOCAL` flag in that word. It does not support any of the other `termio(7I)` device control functions specified by flags in the `c_cflag` word of the `termios` structure and by the `IGNBRK`, `IGNPAR`, `PARMRK`, or `INPCK` flags in the `c_iflag` word of the `termios` structure, as these functions apply only to asynchronous serial ports. All other `termio(7I)` functions must be performed by STREAMS modules pushed atop the driver; when a slave device is opened, the `ldterm(7M)` and `ttcompat(7M)` STREAMS modules are automatically pushed on top of the stream, providing the standard `termio(7I)` interface.

Instead of having a hardware interface and associated hardware that supports the terminal functions, the functions are implemented by another process manipulating the controller device of the pseudo-terminal.

The controller and the slave devices of the pseudo-terminal are tightly connected. Any data written on the controller device is given to the slave device as input, as though it had been received from a hardware interface. Any data written on the slave terminal can be read from the controller device (rather than being transmitted from a UAR).

By default, 48 pseudo-terminal pairs are configured as follows:

```
/dev/pty[p-r][0-9a-f] controller devices
/dev/tty[p-r][0-9a-f] slave devices
```

ioctls The standard set of `termio` `ioctls` are supported by the slave device. None of the bits in the `c_cflag` word have any effect on the pseudo-terminal, except that if the baud rate is set to `B0`, it will appear to the process on the controller device as if the last process on the slave device had closed the line; thus, setting the baud rate to `B0` has the effect of "hanging up" the pseudo-terminal, just as it has the effect of "hanging up" a real terminal.

There is no notion of "parity" on a pseudo-terminal, so none of the flags in the `c_iflag` word that control the processing of parity errors have any effect. Similarly, there is no notion of a "break", so none of the flags that control the processing of breaks, and none of the `ioctls` that generate breaks, have any effect.

Input flow control is automatically performed; a process that attempts to write to the controller device will be blocked if too much unconsumed data is buffered on the slave device. The input flow control provided by the `IXOFF` flag in the `c_iflag` word is not supported.

The delays specified in the `c_oflag` word are not supported.

As there are no modems involved in a pseudo-terminal, the `ioctls` that return or alter the state of modem control lines are silently ignored.

A few special `ioctl`s are provided on the controller devices of pseudo-terminals to provide the functionality needed by applications programs to emulate real hardware interfaces:

`TIOCSTOP` The argument is ignored. Output to the pseudo-terminal is suspended, as if a `STOP` character had been typed.

`TIOCSTART` The argument is ignored. Output to the pseudo-terminal is restarted, as if a `START` character had been typed.

`TIOCPKT` The argument is a pointer to an `int`. If the value of the `int` is non-zero, *packet* mode is enabled; if the value of the `int` is zero, packet mode is disabled. When a pseudo-terminal is in packet mode, each subsequent `read(2)` from the controller device will return data written on the slave device preceded by a zero byte (symbolically defined as `TIOCPKT_DATA`), or a single byte reflecting control status information. In the latter case, the byte is an inclusive-or of zero or more of the bits:

- `TIOCPKT_FLUSHREAD` whenever the read queue for the terminal is flushed.
- `TIOCPKT_FLUSHWRITE` whenever the write queue for the terminal is flushed.
- `TIOCPKT_STOP` whenever output to the terminal is stopped using ^S.
- `TIOCPKT_START` whenever output to the terminal is restarted.
- `TIOCPKT_DOSTOP` whenever XON/XOFF flow control is enabled after being disabled; it is considered "enabled" when the `IXON` flag in the `c_iflag` word is set, the `VSTOP` member of the `c_cc` array is ^S and the `VSTART` member of the `c_cc` array is ^Q.
- `TIOCPKT_NOSTOP` whenever XON/XOFF flow control is disabled after being enabled.

`TIOCREMOTE` The argument is a pointer to an `int`. If the value of the `int` is non-zero, *remote* mode is enabled; if the value of the `int` is zero, remote mode is disabled. This mode can be enabled or disabled independently of packet mode. When a pseudo-terminal is in remote mode, input to the slave device of the pseudo-terminal is flow controlled and not input edited (regardless of the mode the slave side of the pseudo-terminal). Each write to the controller device produces a record boundary for the process reading the slave device. In normal usage, a write of data is like the data typed as a line on the terminal; a write of 0 bytes is like typing an `EOF` character. Note: this means that a process writing to a pseudo-terminal controller in *remote* mode must keep track of line boundaries, and write only one line at a time to the controller. If,

for example, it were to buffer up several NEWLINE characters and write them to the controller with one `write()`, it would appear to a process reading from the slave as if a single line containing several NEWLINE characters had been typed (as if, for example, a user had typed the LNEXT character before typing all but the last of those NEWLINE characters). Remote mode can be used when doing remote line editing in a window manager, or whenever flow controlled input is required.

Examples

```
#include <fcntl.h>
#include <sys/termios.h>

int fdm fds;
fdm = open("/dev/ptyp0, O_RDWR);  /* open master */
fds = open("/dev/ttyp0, O_RDWR);  /* open slave */
```

Files

`/dev/pty[p-z][0-9a-f]`	pseudo-terminal controller devices
`/dev/tty[p-z][0-9a-f]`	pseudo-terminal slave devices

See Also `rlogin(1)`, `rlogind(1M)`, `ldterm(7M)`, `termio(7I)`, `ttcompat(7M)`,

Notes It is apparently not possible to send an `EOT` by writing zero bytes in `TIOCREMOTE` mode.

Name qfe – SUNW,qfe Quad Fast-Ethernet device driver

Synopsis `/dev/qfe`

Description The SUNW,qfe Quad Fast-Ethernet driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface, dlpi(7P), over a SUNW,qfe Quad Fast-Ethernet controller. Multiple SUNW,qfe controllers installed within the system are supported by the driver. The `qfe` driver provides basic support for the SUNW,qfe hardware. It is used to handle the SUNW,qfe device. Functions include chip initialization, frame transit and receive, multicast and promiscuous support, and error recovery and reporting.

SUNW,qfe The SUNW,qfe device provides a 100Base-TX networking interface. There are two types of SUNW,qfe device: one supporting Sbus and the other supporting the PCI bus interface. The Sbus SUNW,qfe device uses Sun's `FEPS ASIC`, which provides the Sbus interface and MAC functions. The PCI SUNW,qfe device uses Sun's `PFEX ASIC` to provide the PCI interface and MAC functions. Both connect with the 100Base-TX on-board transceiver, which connects to a `RJ45` connector to provide the Physical layer functions and external connection.

The 100Base-TX standard specifies an "auto-negotiation" protocol to automatically select the mode and speed of operation. The internal transceiver is capable of doing auto-negotiation with the remote-end of the link (link partner) and receives the capabilities of the remote end. It selects the `Highest Common Denominator` mode of operation based on the priorities. It also supports `forced-mode` of operation where the driver can select the mode of operation.

Application Programming Interface The cloning character-special device `/dev/qfe` is used to access all SUNW,qfe controllers installed within the system.

`qfe` and DLPI The `qfe` driver is a "style 2" data link service provider. All `M_PROTO` and `M_PCPROTO` type messages are interpreted as `DLPI` primitives. Valid `DLPI` primitives are defined in `<sys/dlpi.h>`. Refer to dlpi(7P) for more information. An explicit `DL_ATTACH_REQ` message by the user is required to associate the opened stream with a particular device (`ppa`). The `ppa` ID is interpreted as an `unsigned long` data type and indicates the corresponding device instance (unit) number. The driver returns an error (`DL_ERROR_ACK`) if the `ppa` field value does not correspond to a valid device instance number for this system. The device is initialized on first attach and de-initialized (stopped) at last detach.

The values returned by the driver in the `DL_INFO_ACK` primitive in response to the `DL_INFO_REQ` from the user are as follows:

- The maximum SDU is `1500` (`ETHERMTU` - defined in `<sys/ethernet.h>`).
- The minimum SDU is `0`.
- The `dlsap` address length is `8`.
- The MAC type is `DL_ETHER`.

- The sap length values is –2 meaning the physical address component is followed immediately by a 2 byte sap component within the DLSAP address.
- The service mode is DL_CLDLS.
- No optional quality of service (QOS) support is included at present so the QOS fields are 0.
- The provider style is DL_STYLE2.
- The version is DL_VERSION_2.
- The broadcast address value is Ethernet/IEEE broadcast address (0xFFFFFF).

Once in the DL_ATTACHED state, the user must send a DL_BIND_REQ to associate a particular *service access pointer* SAP with the stream. The qfe driver interprets the sap field within the DL_BIND_REQ as an Ethernet "type" therefore valid values for the sap field are in the [0-0xFFFF] range. Only one Ethernet type can be bound to the stream at any time.

If the user selects a sap with a value of 0, the receiver will be in "802.3 mode". All frames received from the media having a "type" field in the range [0-1500] are assumed to be 802.3 frames and are routed up all open streams which are bound to sap value 0. If more than one stream is in "802.3 mode" then the frame will be duplicated and routed up multiple streams as DL_UNITDATA_IND messages.

In transmission, the driver checks the sap field of the DL_BIND_REQ if the sap value is 0, and if the destination type field is in the range [0-1500]. If either is true, the driver computes the length of the message, not including initial M_PROTO mblk (message block), of all subsequent DL_UNITDATA_REQ messages and transmits 802.3 frames that have this value in the MAC frame header length field.

The qfe driver DLSAP address format consists of the 6 byte physical (Ethernet) address component followed immediately by the 2 byte sap (type) component producing an 8 byte DLSAP address. Applications should *not* hardcode to this particular implementation-specific DLSAP address format but use information returned in the DL_INFO_ACK primitive to compose and decompose DLSAP addresses. The sap length, full DLSAP length, and sap/physical ordering are included within the DL_INFO_ACK. The physical address length can be computed by subtracting the sap length from the full DLSAP address length or by issuing the DL_PHYS_ADDR_REQ to obtain the current physical address associated with the stream.

Once in the DL_BOUND state, the user may transmit frames on the Ethernet by sending DL_UNITDATA_REQ messages to the qfe driver. The qfe driver will route received Ethernet frames up all those open and bound streams having a sap which matches the Ethernet type as DL_UNITDATA_IND messages. Received Ethernet frames are duplicated and routed up multiple open streams if necessary. The DLSAP address contained within the DL_UNITDATA_REQ and DL_UNITDATA_IND messages consists of both the sap (type) and physical (Ethernet) components.

In addition to the mandatory connectionless DLPI message set the driver also supports the following primitives.

qfe Primitives

The DL_ENABMULTI_REQ and DL_DISABMULTI_REQ primitives enable or disable reception of individual multicast group addresses. A set of multicast addresses may be iteratively created and modified on a per-stream basis using these primitives. The driver accepts these primitives in any state following DL_ATTACHED.

The DL_PROMISCON_REQ and DL_PROMISCOFF_REQ primitives with the DL_PROMISC_PHYS flag set in the dl_level field enables or disables reception of all frames on the media ("promiscuous mode"), including frames generated by the local host.

When used with the DL_PROMISC_SAP flag set this enables or disables reception of all sap (Ethernet type) values. When used with the DL_PROMISC_MULTI flag set this enables or disables reception of all multicast group addresses. The effect of each is always on a per-stream basis and independent of the other sap and physical level configurations on this stream or other streams.

The DL_PHYS_ADDR_REQ primitive returns the 6 octet Ethernet address currently associated (attached) to the stream in the DL_PHYS_ADDR_ACK primitive. This primitive is valid only in states following a successful DL_ATTACH_REQ.

The DL_SET_PHYS_ADDR_REQ primitive changes the 6 octet Ethernet address currently associated (attached) to this stream. The credentials of the process which originally opened this stream must be root. Otherwise EPERM is returned in the DL_ERROR_ACK. This primitive is destructive in that it affects all other current and future streams attached to this device. An M_ERROR is sent up all other streams attached to this device when this primitive is successful on this stream. Once changed, all streams subsequently opened and attached to this device will obtain this new physical address. Once changed, the physical address will remain until this primitive is used to change the physical address again or the system is rebooted, whichever comes first.

qfe Driver

By default, the qfe driver performs "auto-negotiation" to select the mode and speed of the link.

The link can be in one of the four following modes:

- 100 Mbps, full-duplex
- 100 Mbps, half-duplex
- 10 Mbps, full-duplex
- 10 Mbps, half-duplex

These speeds and modes are described in the 100Base-TX standard.

The auto–negotiation protocol automatically selects:

- Operation mode (half-duplex or full-duplex)
- Speed (100 Mbps or 10 Mbps)

The auto–negotiation protocol does the following:

- Gets all the modes of operation supported by the Link Partner
- Advertises its capabilities to the Link Partner
- Selects the highest common denominator mode of operation based on the priorities.
- The highest priority is given to the 100 Mbps, full-duplex; lowest priority is given to 10 Mbps, half-duplex.

The *100Base-TX transceiver* is capable of all of the operating speeds and modes listed above. By default, auto-negotiation is used to select the speed and the mode of the link and the common mode of operation with the link partner.

Sometimes, the user may want to select the speed and mode of the link. The SUNW,qfe device supports programmable "`IPG`" (Inter-Packet Gap) parameters `ipg1` and `ipg2`. By default, the driver sets `ipg1` to 8 `byte-times` and `ipg2` to 4 `byte-times` (which are the standard values). Sometimes, the user may want to alter these values depending on whether the driver supports 10 Mbps or 100 Mpbs and accordingly, `IPG` will be set to 9.6 or 0.96 microseconds.

`qfe` Parameter List

The qfe driver provides for setting and getting various parameters for the SUNW,qfe device. The parameter list includes:

- current transceiver status
- current link status
- inter-packet gap
- local transceiver capabilities
- link partner capabilities

The local transceiver has two sets of capabilities: one set reflects the capabilities of the hardware, which are read-only (RO) parameters, and the second set, which reflects the values chosen by the user, is used in speed selection. There are read/write (RW) capabilities. At boot time, these two sets of capabilities will be the same. The Link Partner capabilities are also read-only parameters because the current default value of these parameters can only be read and cannot be modified.

Files

`/dev/qfe`	`qfe` special character device
`/kernel/drv/qfe.conf`	system wide default device driver properties

See Also `ndd(1M)`, `netstat(1M)`, `driver.conf(4)`, `dlpi(7P)`

Name qlc – ISP2200, ISP2300, and SP212 Family Fibre Channel host bus adapter driver.

Synopsis `SUNW,qlc`

Description The `qlc` host bus adapter driver is a Sun Fibre Channel transport layer-compliant nexus driver for the Qlogic ISP2200, ISP2200A, ISP2310, ISP2312, and SP212 adapters. These adapters support Fibre Channel SCSI and IP Protocols, FC-AL public loop profile, point-to-point fabric connection and Fibre Channel service classes two and three (see NOTES section below).

The `qlc` driver interfaces with the Sun Fibre Channel transport layer to support the standard functions provided by the SCSA interface. It supports auto request sense and tagged queueing by default. The driver requires that all devices have unique hard addresses in private loop configurations. Devices with conflicting hard addresses are not accessible.

Files

`/kernel/drv/qlc`	32–bit ELF kernel module (x86)
`/kernel/drv/amd64/qlc`	64-bit ELF kernel module (x86)
`/kernel/drv/sparcv9/qlc`	64-bit ELF kernel module (SPARC)
`/kernel/drv/qlc.conf`	Driver configuration file

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86
Availability	SUNWqlc

See Also `prtconf(1M)`, `driver.conf(4)`, `fcp(7D)`, `fp(7d)`

Writing Device Drivers

ANSI X3.230:1994, Fibre Channel Physical Signaling (FC-PH)

Project 1134-D, Fibre Channel Generic Services (FC-GS-2)

ANSI X3.269-1996, Fibre Channel Arbitrated Loop (FC-AL)

ANSI X3.270-1996, Fibre Channel Protocol for SCSI (FCP-SCSI)

ANSI X3.270-1996, SCSI-3 Architecture Model (SAM)

Fibre Channel Private Loop SCSI Direct Attach (FC-PLDA)

Fabric Loop Attachment (FC-FLA)

ISP2200 Firmware Interface Specification, QLogic Corporation

ISP2300 Series Firmware Specification, QLogic Corporation

Notes SP-212–based host bus adapters (including QLA-210) are supported on x86 platforms only and are limited to a maximum of 8 targets in fabric and sixteen targets in local loop topology. FL topology is not supported with the SP-212–based host bus adapter.

Name qlge – 10 Gigabit Ethernet driver for QLogic QLE81XX Converged Network Adapter Family

Synopsis `SUNW,qlge`

Description The `qlge` 10 Gigabit Ethernet driver is a multi-threaded, Loadable, clonable, GLDv3-based driver. The `qlge` driver provides basic support including chip initialization, auto-negotiation, packet transmit and receive, Jumbo Frame, promiscuous and multicast support, 802.3x Standard Ethernet Flow Control and Class Based Flow Control (CBFC), Checksum Offload, Large Send Offload (LSO).

Configuration The `qlge` driver is managed by the `dladm(1M)`command line utility, which allows VLANs to be defined on top of qlge instances and for qlge instances to be aggregated. See `dladm(1M)` for details.

Users can also modify `qlge.conf` to change default settings, like `mtu`, flow control mode, and so forth.

Files

`/kernel/drv/qlge`	32-bit ELF kernel module, x86
`/kernel/drv/amd64/qlge`	64-bit ELF kernel module, x86
`/kernel/drv/sparcv9/qlge`	64-bit ELF kernel module, SPARC
`/kernel/drv/qlge.conf`	Driver configuration file

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SPARC, x86
Availability	SUNWqlc

See Also `dladm(1M)`, `prtconf(1M)`, `attributes(5)`

Writing Device Drivers

Name quotactl – manipulate disk quotas

Synopsis

```
#include <sys/fs/ufs_quota.h>
int ioctl(int fd, Q_QUOTACTL, struct quotcl *qp)
```

Description This `ioctl()` call manipulates disk quotas. *fd* is the file descriptor returned by the `open()` system call after opening the `quotas` file (located in the root directory of the filesystem running quotas.) `Q_QUOTACTL` is defined in `/usr/include/sys/fs/ufs_quota.h`. *qp* is the address of the `quotctl` structure which is defined as

```
struct quotctl {
  int op;
  uid_t uid;
  caddr_t addr;
};
```

op indicates an operation to be applied to the user ID *uid*. (See below.) *addr* is the address of an optional, command specific, data structure which is copied in or out of the system. The interpretation of *addr* is given with each value of *op* below.

`Q_QUOTAON` Turn on quotas for a file system. *addr* points to the full pathname of the `quotas` file. *uid* is ignored. It is recommended that *uid* have the value of `0`. This call is restricted to the super-user.

`Q_QUOTAOFF` Turn off quotas for a file system. *addr* and *uid* are ignored. It is recommended that *addr* have the value of `NULL` and *uid* have the value of `0`. This call is restricted to the super-user.

`Q_GETQUOTA` Get disk quota limits and current usage for user *uid*. *addr* is a pointer to a `dqblk` structure (defined in `<sys/fs/ufs_quota.h>`). Only the super-user may get the quotas of a user other than himself.

`Q_SETQUOTA` Set disk quota limits and current usage for user *uid*. *addr* is a pointer to a `dqblk` structure (defined in `sys/fs/ufs_quota.h`). This call is restricted to the super-user.

`Q_SETQLIM` Set disk quota limits for user *uid*. *addr* is a pointer to a `dqblk` structure (defined in `sys/fs/ufs_quota.h`). This call is restricted to the super-user.

`Q_SYNC` Update the on-disk copy of quota usages for this file system. *addr* and *uid* are ignored.

`Q_ALLSYNC` Update the on-disk copy of quota usages for all file systems with active quotas. *addr* and *uid* are ignored.

Return Values This `ioctl()` returns:

`0` on success.

`−1` on failure and sets `errno` to indicate the error.

Errors

EFAULT	*addr* is invalid.
EINVAL	The kernel has not been compiled with the QUOTA option. *op* is invalid.
ENOENT	The quotas file specified by *addr* does not exist.
EPERM	The call is privileged and the calling process did not assert {PRIV_SYS_MOUNT} in the effective set.
ESRCH	No disk quota is found for the indicated user. Quotas have not been turned on for this file system.
EUSERS	The quota table is full.

If *op* is Q_QUOTAON, ioctl() may set errno to:

EACCES	The quota file pointed to by *addr* exists but is not a regular file. The quota file pointed to by *addr* exists but is not on the file system pointed to by *special*.
EIO	Internal I/O error while attempting to read the quotas file pointed to by *addr*.

Files

/usr/include/sys/fs/ufs_quota.h	quota-related structure/function definitions and defines

See Also quota(1M), quotacheck(1M), quotaon(1M), getrlimit(2), mount(2)

Bugs There should be some way to integrate this call with the resource limit interface provided by setrlimit() and getrlimit(2).

This call is incompatible with Melbourne quotas.

Name qus – Qlogic Ultra3 SCSI ISP10160 Host Bus Adapter Driver

Synopsis

```
pci@pci-slot/scsi@4 - Scsi bus-1

pci@pci-slot/scsi@5 - Scsi bus-2
```

Description The ISP10160 host bus adapter is a SCSA-compliant nexus driver that supports Qlogic ISP10160 SCSI chips on the PCI bus. The ISP10160 is an intelligent SCSI host bus adapter chip that reduces the amount of CPU overhead used in a SCSI transfer.

The `qus` driver supports the standard functions provided by the SCSA interface. The driver supports tagged and untagged queuing, fast, fast-20, fast-40, fast-80, and auto request sense, but does not support linked commands.

Configuration You configure the `qus` driver by defining properties in `qus.conf` which override the global SCSI settings. Supported properties are `scsi-options`, `target<`*n*`>-scsi-options`, `scsi-reset-delay`, `scsi-watchdog-tick`, `scsi-tag-age-limit`, `scsi- initiator-id`, and `scsi-selection-timeout`.

`target<`*n*`>-scsi-options` overrides the `scsi-options` property value for target<*n*>.<*n*> is a hex value that can vary from *0* to *f*. Refer to `scsi_hba_attach(9F)` for details.

Examples Example 1: SCSI Options:

Create a file called `/kernel/drv/qus.conf`, then add the following line to disable tagged queuing, fast SCSI, and Wide mode for all `qus` instances:

```
scsi-options=0x78;
```

To disable an option for a specific ISP10160 (see `driver.conf(4)`), do the following:

```
name="qus" parent="/pci@1f,2000/pci@1"
  unit-address="4"
  scsi-options=0x178
  target3-scsi-options=0x58 scsi-initiator-id=6;
```

The default initiator ID in OBP is 7 and that the change to ID 6 occurs at attach time. You might prefer to change the initiator ID in OBP.

Example 1 sets `scsi-options` for target 3 to 0x58 and all other targets on this SCSI bus to 0x178.

To determine the physical pathname of the parent, use the `/devices` tree or follow the link of the logical device name:

```
example# ls -l /dev/rdsk/c2t0d0s0
     lrwxrwxrwx   1 root   root  76 Aug 22 13:29 /dev/rdsk/c2t0d0
     s0 -> ../../devices/pci@1f,2000/pci@1/scsi@5/sd@0,0:a,raw
```

To determine the hardware property values, use the output of prtconf(1M) with the -v option:

```
pci, instance #0
     Driver properties:
         name='device_type' type=string items=1 dev=none
             value='pci'
     Hardware properties:
         name='ranges' type=int items=8
             value=82000000.00000000.00100000.82000000.00000000.00100000./
                   00000000.00100000
         name='latency-timer' type=int items=1
                   value=00000040
         name='cache-line-size' type=int items=1
                   value=00000010
     scsi, instance #0
          Driver properties:
              name='scsi-selection-timeout' type=int items=1 dev=(249,0)
                   value=000000fa
              name='scsi-options' type=int items=1 dev=(249,0)
                   value=00107ff8
              name='scsi-watchdog-tick' type=int items=1 dev=(249,0)
                   value=0000000a
              name='scsi-tag-age-limit' type=int items=1 dev=(249,0)
                   value=00000002
              name='scsi-reset-delay' type=int items=1 dev=(249,0)
                   value=00000bb8
          Hardware properties:
              name='cache-line-size' type=int items=1
                   value=00000010
          sd (driver not attached)
          st (driver not attached)
      scsi, instance #1
          Driver properties:
              name='scsi-selection-timeout' type=int items=1 dev=(249,0)
                  value=000000fa
              name='scsi-options' type=int items=1 dev=(249,0)
                  value=00107ff8
              name='scsi-watchdog-tick' type=int items=1 dev=(249,0)
                  value=0000000a
              name='scsi-tag-age-limit' type=int items=1 dev=(249,0)
                  value=00000002
              name='scsi-reset-delay' type=int items=1 dev=(249,0)
                  value=00000bb8
          Hardware properties:
              name='cache-line-size' type=int items=1
                  value=00000010
```

```
          sd (driver not attached)
          st (driver not attached)
```

Example 2: ISP10160 Properties

The qus driver exports properties indicating (per target) the negotiated transfer speed (target<*n*>-sync-speed), whether tagged queuing has been enabled (target<*n*>-TQ), and whether the wide data transfer has been negotiated (target<*n*>-wide). The `sync-speed` property value is the data transfer rate in KB/sec. The `target-TQ` and `target-wide` properties have no value. The existence of these properties indicate that tagged queuing or wide transfer is enabled. Refer to `prtconf(1M)` (verbose option) for information on qus properties.

```
scsi, instance #1
    Driver properties:
        name='target2-wide' type=boolean dev=none
        name='target2-TQ' type=boolean dev=none
        name='target2-sync-speed' type=int items=1 dev=none
            value=00027100
        name='target0-wide' type=boolean dev=none
        name='target0-TQ' type=boolean dev=none
        name='target0-sync-speed' type=int items=1 dev=none
            value=00027100
```

To determine the physical pathname of the parent, use the `/devices` tree or follow the link of the logical deiice name.

To set `scsi-options` more specifically per device type, add the following line to the `/kernel/drv/qus.conf` file:

```
device-type-scsi-options-list =
     "SEAGATE ST32550W", "seagate-scsi-options" ;
     seagate-scsi-options = 0x58;
```

All devices of this specific disk type have `scsi-options` set to 0x58.

`scsi-options` specified per target ID has the highest precedence, followed by `scsi-options` per device type. Global (for all qus instances) `scsi-options` per bus has the lowest precedence.

You must reboot your system for the specified `scsi-options` to take effect.

Example 3: Driver Capabilities

To enable some driver features, the target driver must set capabilities in the qus driver. The target driver can query and modify the following capabilities: synchronous, tagged-qing, wide-xfer, auto-rqsense, qfull-retries, qfull-retry-interval. All other capabilities are query only.

By default, tagged-qing, auto-rqsense, and wide-xfer capabilities are disabled, while disconnect, synchronous, and untagged-qing are enabled. These capabilities can have binary values (0 or 1) only. The default value for both qfull-retries and qfull-retry-interval is 10. The qfull-retries capability is a uchar_t (0 to 255), while qfull-retry-interval is a ushort_t (0 to 65535).

The target driver must enable tagged-qing and wide-xfer explicitly. The untagged-qing capability is always enabled and its value cannot be modified due to the `qus` driver's ability to queue commands even when tagged-qing is disabled.

When a conflict occurs between the value of `scsi-options` and a capability, the value set in `scsi-options` prevails. Only `whom != 0` is supported in the `scsi_ifsetcap(9F)` call.

Refer to `scsi_ifsetcap(9F)` and `scsi_ifgetcap(9F)` for details.

Files

`/kernel/drv/sparcv9/qus`	64–bitELF kernel module
`/kernel/drv/qus.conf`	Configuration file

Attributes

See `attributes(5)` for a description of the following attribute:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC

See Also

`prtconf(1M)`, `driver.conf(4)` , `attributes(5)`, `scsi_abort(9F)`, `scsi_hba_attach(9F)`, `scsi_ifgetcap(9F)`, `scsi_ifsetcap(9F)`, `scsi_reset(9F)`, `scsi_transport(9F)`, `scsi_device(9S)`, `scsi_extended_sense(9S)`, `scsi_inquiry(9S)`, `scsi_pkt(9S)`

Writing Device Drivers

ANSI Small Computer System Interface-2 (SCSI-2)

SCSI Parallel Interface-3 (SPI-3)

QLogic Corporation, ISP1080/1280/10160A/12160A Initiator Firmware Interface Specification

QLogic Corporation, ISP10160A/12160A Technical Manual

QLogic Corporation, ISP1080 Technical Manual

QLogic Corporation, ISP1280 Technical Manual

Diagnostics

In addition to being logged, the following messages can appear on the system console.

The first set of messages shown below can be displayed while the qus driver is first trying to attach. All messages in the first set indicate that the `qus` driver was unable to attach. Each message is preceded by "`qus`<*number*>)", where "<*number*>" is the instance number of the JASPER Host Bus Adapter.

Device in slave-only slot, unused	The SBus device has been placed in a slave-only slot and are not accessible; move to non- slave-only SBus slot.
Device is using a hilevel intr, unused	The device was configured with an interrupt level that cannot be used with the qus driver. Check the device.
Failed to alloc soft state	Driver is unable to allocate space for the internal state structure. Driver did not attach to device; SCSI devices are inaccessible.
Bad soft state	Driver requested an invalid internal state structure. Driver did not attach to device; SCSI devices are inaccessible.
Can't reload firmware: failing attach/resume	Driver is unable to reload firmware; check for bad hardware. Driver did not attach to device; SCSI devices are inaccessible.
Can't reset interface during attach/resume	Driver is unable to reset the hardware. Driver did not attach to device; SCSI devices are inaccessible.
Cannot find PCI device-id	Driver is unable to find PCI device-id. Driver did not attach to device; SCSI devices are inaccessible.
Unable to support ISP chip	Driver is unable to support the ISP chip, which is in the interface. Driver did not attach to device; SCSI devices are inaccessible.
Unable to map pci config registers	Driver is unable to map device registers. Check for bad hardware. Driver did not attach to device. SCSI devices are inaccessible.
Unable to attach: check for hardware problem	Driver is unable to attach to the hardware. Driver did not attach to device; SCSI devices are inaccessible.

The following messages can be displayed at any time and are printed with the full device pathname followed by the shorter form described above.

Firmware (<actual-size>) should be <allowed_size> bytes
Firmware id(<id>) verification failed
Firmware length too short
Firmware checksum incorrect

Can't find firmware rev. string
reset/init ISP chip failed
reset/init ISP chip failed
Load ram failed
Start firmware mailbox command failed
Can't set clock rate
Can't get RAM info
These messages indicate a firmware download failure and possible corruption of the firmware. Check the ISP driver.

Chip reset timeout
ISP chip failed to reset in the time allocated. Potential hardware problem.

Bad request pkt payload
The ISP Firmware rejected the packet, indicating that the packet was set up incorrectly. As a result, the `qus` driver calls the target completion routine with the reason of CMD_TRAN_ERR set in the `scsi_pkt`. To correctly set up the packet, check the target driver.

Bad request pkt header
The ISP Firmware rejected the packet because the packet was set up incorrectly. As a result, the `qus` driver calls the target completion routine with the reason of CMD_TRAN_ERR set in the `scsi_pkt`. To correctly set up the packet, check the target driver.

Target synch. rate reduced. tgt <target-id>
Target<target-id> reducing transfer rate
These messages indicate that the target is reducing its transfer rate. Reboot the system to obtain the maximum transfer rate.

Failed to Get Features
Chip reset detected
These messages indicate a possible ISP chip failure. Driver attempts to recover from this condition by reloading and restarting the firmware.

Interface going offline
Although all driver recovery procedure are completed, the interface did not come online and might need replacement.

SCSI Cable/Connection problem
The SCSI cable is faulty or is connected improperly.

Hardware/Firmware error
The ISP chip encountered a firmware error that is probably due to a faulty SCSI cable or improper cable connection. As a result, the `qus` driver attempts to do error recovery by resetting the chip.

Received unexpected SCSI Reset
The ISP chip received an unexpected SCSI Reset and has initiated its own internal error recovery, which returns `scsi_pkt` with reason set to CMD_RESET.

Fatal error, resetting interface
: The `qus` driver is performing error recovery. As a result, all outstanding commands that have been transported to the `qus` driver is completed by way of the `scsi_pkt` completion routine in the target driver with reason of CMD_RESET and status of STAT_BUS_RESET set in `scsi_pkt`.

Phase skipped: Command completed with good status — Potential data path failure:
: Possible SCSI cable failure. Driver might still be able to communicate with the target and continue sending commands. Leaving the system in the same state triggers this message once per hour. System I/O throughput might be reduced. The SCSI might need to be replaced.

LVD Error detected
SCSI Cable/Connection problem
Bus Not Terminated
f/w initiated BDR fails
: These messages indicate a possible cable failure. Outstanding commands are returned with CMD_RESET or CMD_TRAN_ERR set in `scsi_pkt`.

Name radeon – DRI (Direct Rendering Infrastructure)-compliant kernel driver providing 3-dimensional graphic hardware acceleration support.

Description The `radeon` driver is a DRI-compliant kernel driver that provides graphics hardware acceleration support. DRI is a framework for coordinating OS kernel, 3D graphics hardware, X window system and OpenGL applications.

The `radeon` driver currently supports certain low-end ATI radeon graphics cards, including Radeon X700.

Files

`/kernel/drv/radeon`	32-bit ELF kernel module (x86).
`/kernel/drv/amd64/radeon`	64-bit ELF kernel module (x86).

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWdrmr
Architecture	x86
Interface Stability	Committed

See Also

`/usr/X11/share/man/man1/Xserver.1`

`/usr/X11/share/man/man1/Xorg.1`

`/usr/X11/share/man/man5/X11.5`

`attributes(5)`

Name ral – Ralink RT2500 802.11b/g Wireless driver

Description The `ral` *802.11b/g* wireless NIC driver is a multi-threaded, loadable, clonable, GLDv3-based STREAMS driver supporting Ralink RT2500 chipset-based NIC's.

Configuration The `ral` driver performs auto-negotiation to determine the data rate and mode. Supported *802.11b* data rates are 1, 2, 5.5 and 11 Mbits/sec. Supported *802.11g* data rates are 1, 2, 5.5, 6, 9, 11, 12, 18, 24, 36, 48 and 54 Mbits/sec. The `ral` driver supports only BSS networks (also known as "ap" or "infrastructure" networks) and "open"(or "open-system") or "shared system" authentication.

Files `/dev/ral*`
Special character device.

/kernel/drv/ral
32-bit ELF kernel module (x86).

`/kernel/drv/amd64/ral`
64-bit ELF kernel module (x86).

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWralink
Interface stability	Committed

See Also `dladm(1M)`, `wificonfig(1M)`, `attributes(5)`, `gld(7D)`, `dlpi(7P)`

802.11 - Wireless LAN Media Access Control and Physical Layer Specification — IEEE, 2001

Name ramdisk – RAM disk device driver

Synopsis `ramdisk@0:`*diskname*

Description The `ramdisk` driver supports numerous ramdisk devices that are created by the system during the boot process (see `boot(1M)`) or during normal system operation (see `ramdiskadm(1M)` for more information).

Device Special Files Each ramdisk can be accessed either as a block device or as a raw device. When accessed as a block device, the normal buffering mechanism is used when reading from and writing to the device, without regard to physical disk records. Accessing the ramdisk as a raw device enables direct transmission between the disk and the read or write buffer. A single read or write call usually results in a single I/O operation, meaning that raw I/O is more efficient when many bytes are transmitted. You can find block files names in `/dev/ramdisk`. Raw file names are found in `/dev/rramdisk`.

There are no alignment or length restrictions on I/O requests to either block or character devices.

Errors

`EFAULT` The argument features a bad address.

`EINVAL` Invalid argument. EIO. An I/O error occurred.

`EPERM` Cannot create or delete a ramdisk without write permission on `/dev/ramdiskctl`.

`ENOTTY` The device does not support the requested ioctl function.

`ENXIO` The device did not exist during opening.

`EBUSY` Cannot exclusively open `/dev/ramdiskctl`. One or more ramdisks are still open.

`EEXIST` A ramdisk with the indicated name already exists.

`EAGAIN` Cannot allocate resource for ramdisk. Try again later.

Files

`/dev/ramdisk/diskname`	Block device for ramdisk named *diskname*.
`/dev/rramdisk/diskname`	Raw device for ramdisk name *diskname*
`/kernel/drv/ramdisk`	32-bit driver
`/kernel/drv/ramdisk.conf`	Driver configuration file. (Do not alter).
`/kernel/drv/sparcv9/ramdisk`	64-bit driver

Attributes See `attributes(5)` for descriptions of the following attribute:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

See Also ramdiskadm(1M), fsck(1M), fstyp(1M), mount(1M), newfs(1M), driver.conf(4), filesystem(5), dkio(7I)

Notes The percentage of available physical memory that can be allocated to ramdisks is constrained by the variable `rd_percent_physmem`. You can tune the `rd_percent_physmem` variable in `/etc/system`. By default, the percentage of available physical memory that can be allocated to ramdisks is fixed at 25%.

A ramdisk may not be the best possible use of system memory. Accordingly, use ramdisks only when absolutely necessary.

Name random, urandom – Strong random number generator device

Synopsis

```
/dev/random

/dev/urandom
```

Description The `/dev/random` and `/dev/urandom` files are special files that are a source for random bytes generated by the kernel random number generator device. The `/dev/random` and `/dev/urandom` files are suitable for applications requiring high quality random numbers for cryptographic purposes.

The generator device produces random numbers from data and devices available to the kernel and estimates the amount of randomness (or "entropy") collected from these sources. The entropy level determines the amount of high quality random numbers that are produced at a given time.

Applications retrieve random bytes by reading `/dev/random` or `/dev/urandom`. The `/dev/random` interface returns random bytes only when sufficient amount of entropy has been collected. If there is no entropy to produce the requested number of bytes, `/dev/random` blocks until more entropy can be obtained. Non-blocking I/O mode can be used to disable the blocking behavior. The `/dev/random` interface also supports `poll(2)`. Note that using `poll(2)` will not increase the speed at which random numbers can be read.

Bytes retrieved from `/dev/random` provide the highest quality random numbers produced by the generator, and can be used to generate long term keys and other high value keying material.

The `/dev/urandom` interface returns bytes regardless of the amount of entropy available. It does not block on a read request due to lack of entropy. While bytes produced by the `/dev/urandom` interface are of lower quality than bytes produced by `/dev/random`, they are nonetheless suitable for less demanding and shorter term cryptographic uses such as short term session keys, paddings, and challenge strings.

Data can be written to `/dev/random` and `/dev/urandom`. Data written to either special file is added to the generator's internal state. Data that is difficult to predict by other users may contribute randomness to the generator state and help improve the quality of future generated random numbers.

`/dev/random` collects entropy from providers that are registered with the kernel-level cryptographic framework and implement random number generation routines. The `cryptoadm(1M)` utility allows an administrator to configure which providers will be used with `/dev/random`.

Errors

`EAGAIN`	`O_NDELAY` or `O_NONBLOCK` was set and no random bytes are available for reading from `/dev/random`.
`EINTR`	A signal was caught while reading and no data was transferred.
`ENOXIO`	`open(2)` request failed on `/dev/random` because no entropy provider is available.

Files `/dev/random`

`/dev/urandom`

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsr
Interface Stability	Evolving

See Also `cryptoadm(1M)`, `open(2)`, `poll(2)`, `attributes(5)`

Notes `/dev/random` can be configured to use only the hardware-based providers registered with the kernel-level cryptographic framework by disabling the software-based provider using `cryptoadm(1M)`. You can also use `cryptoadm(1M)` to obtain the name of the software-based provider.

Because no entropy is available, disabling all randomness providers causes `read(2)` and `poll(2)` on `/dev/random` to block indefinitely and results in a warning message being logged and displayed on the system console. However, `read(2)` and `poll(2)` on `/dev/urandom` continue to work in this case.

An implementation of the `/dev/random` and `/dev/urandom` kernel-based random number generator first appeared in Linux 1.3.30.

A `/dev/random` interface for Solaris first appeared as part of the CryptoRand implementation.

Name rarp, RARP – Reverse address resolution protocol

Description You use the RARP protocol to map dynamically between the Internet Protocol (IP) and network interface MAC addresses. RARP is often used to boot a Solaris client. RARP clients include the SPARC boot PROM, x86 boot floppy, SunOS kernel, and `ifconfig(1M)`. `in.rarpd(1M)` provides the server-side implementation.

RARP request timeout behavior in application-layer clients is governed by the `/etc/inet/rarp` default file. To tune the number of retries an application attempts before giving up, set the `RARP_RETRIES` variable in `/etc/inet/rarp`. If the file is not present or `RARP_RETRIES` is not initialized within it, applications retry a maximum of five times with a eight second wait between retries.

Files `/etc/inet/rarp`

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability (protocol)	Standard
Interface Stability (defaults file)	Unstable
Interface Stability (RARP_RETRIES)	Unstable

See Also `ifconfig(1M)`, `in.rarpd(1M)`, `arp(7P)`

Reverse Address Resolution Protocol RFC 903. June, 1984 R. Finlayson, T. Mann, J.C. Mogul, M. Theimer

Name rge – Realtek Gigabit/Fast Ethernet Network Adapter driver

Synopsis `/dev/rge`

Description The `rge` Gigabit/Fast Ethernet driver is a multi-threaded, loadable, clonable, GLD-based STREAMS driver supporting the Data Link Provider Interface, `dlpi(7P)`, on the Realtek Gigabit/Fast Ethernet Network Adapter.

The `rge` driver functions includes controller initialization, frame transmit and receive, promiscuous and multicast support, and error recovery and reporting.

Application Programming Interface The cloning, character-special device `/dev/rge` is used to access all Realtek Gigabit/Fast Ethernet devices installed within the system.

The `rge` driver is managed by the `dladm(1M)` command line utility, which allows VLANs to be defined on top of `rge` instances and for `rge` instances to be aggregated. See `dladm(1M)` for more details.

The values returned by the driver in the DL_INFO_ACK primitive in response to the DL_INFO_REQ are as follows:

- Maximum SDU (with jumbo frame) is `7000`.
- Minimum SDU is `0`.
- DSLAP address length is `8` bytes.
- MAC type is `DL_ETHER`.
- SAP length value is -2, meaning the physical address component is followed immediately by a 2-byte sap component within the DLSAP address.
- Broadcast address value is Ethernet/IEEE broadcast address (FF:FF:FF:FF:FF:FF).

Once in the DL_ATTACHED state, you must send a DL_BIND_REQ to associate a particular Service Access Point (SAP) with the stream.

Configuration By default, the `rge` driver performs auto-negotiation to select the link speed and mode. Link speed and mode can be any one of the following:

- 1000 Mbps, full-duplex
- 100 Mbps, full-duplex
- 100 Mbps, half-duplex
- 10 Mbps, full-duplex
- 10 Mbps, half-duplex

Alternatively, you can set the capabilities advertised by the `rge` device using `ndd(1M)`. The driver supports a number of parameters whose names begin with *adv_*. Each of these parameters contains a boolean value that determines if the device advertises that mode of operation. The *adv_pause_cap* indicates if half/full duplex pause is advertised to link partner. You can set *adv_asym_pause_cap* to advertise to the link partner that asymmetric pause is desired.

For example, to prevent the device 'rge2' from advertising gigabit capabilities, enter (as super-user):

```
# ndd -set /dev/rge2 adv_1000fdx_cap 0
```

All capabilities default to enabled. Note that changing any capability parameter causes the link to go down while the link partners renegotiate the link speed/duplex using the newly changed capabilities.

You can find the current parameter settings by using `ndd -get`. In addition, the driver exports the current state, speed, duplex setting, and working mode of the link via `ndd` parameters (these are read only and may not be changed). For example, to check link state of device `rge0`:

```
# ndd -get /dev/rge0 link_status
1
# ndd -get /dev/rge0 link_speed
100
# ndd -get /dev/rge0 link_duplex
2
```

The output above indicates that the link is up and running at 100Mbps full-duplex. In addition, the driver exports its working mode by *loop_mode*. If it is set to 0, the loopback mode is disabled.

Files

`/dev/rge*`	Character special device.
`/kernel/drv/rge`	32–bit x86 `rge` driver binary.
`/kernel/drv/amd64/rge`	64–bit x86 `rge` driver binary.
`/kernel/drv/sparcv9/rge`	SPARC `rge` driver binary.

Attributes

See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86

See Also

`dladm(1M)`, `attributes(5)`, `streamio(7I)`, `dlpi(7P)`

Writing Device Drivers

STREAMS Programming Guide

Network Interfaces Programmer's Guide

Name route – kernel packet forwarding database

Synopsis

```
#include <sys/types.h>
#include <sys/socket.h>
#include <net/if.h>
#include <net/route.h>

int socket(PF_ROUTE, SOCK_RAW, int protocol);
```

Description UNIX provides some packet routing facilities. The kernel maintains a routing information database, which is used in selecting the appropriate network interface when transmitting packets.

A user process (or possibly multiple co-operating processes) maintains this database by sending messages over a special kind of socket. This supplants fixed size `ioctl(2)`'s specified in `routing(7P)`. Routing table changes can only be carried out by the superuser.

The operating system might spontaneously emit routing messages in response to external events, such as receipt of a re-direct, or failure to locate a suitable route for a request. The message types are described in greater detail below.

Routing database entries come in two flavors: entries for a specific host, or entries for all hosts on a generic subnetwork (as specified by a bit mask and value under the mask). The effect of wildcard or default route can be achieved by using a mask of all zeros, and there can be hierarchical routes.

When the system is booted and addresses are assigned to the network interfaces, the internet protocol family installs a routing table entry for each interface when it is ready for traffic. Normally the protocol specifies the route through each interface as a *direct* connection to the destination host or network. If the route is direct, the transport layer of a protocol family usually requests the packet be sent to the same host specified in the packet. Otherwise, the interface is requested to address the packet to the gateway listed in the routing entry, that is, the packet is forwarded.

When routing a packet, the kernel attempts to find the most specific route matching the destination. If no entry is found, the destination is declared to be unreachable, and a routing-miss message is generated if there are any listeners on the routing control socket (described below). If there are two different mask and value-under-the-mask pairs that match, the more specific is the one with more bits in the mask. A route to a host is regarded as being supplied with a mask of as many ones as there are bits in the destination.

A wildcard routing entry is specified with a zero destination address value, and a mask of all zeroes. Wildcard routes are used when the system fails to find other routes matching the destination. The combination of wildcard routes and routing redirects can provide an economical mechanism for routing traffic.

One opens the channel for passing routing control messages by using the socket call. There can be more than one routing socket open per system.

Messages are formed by a header followed by a small number of `sockaddrs`, whose length depend on the address family. `sockaddrs` are interpreted by position. An example of a type of message with three addresses might be a CIDR prefix route: Destination, Netmask, and Gateway. The interpretation of which addresses are present is given by a bit mask within the header, and the sequence is least significant to most significant bit within the vector.

Any messages sent to the kernel are returned, and copies are sent to all interested listeners. The kernel provides the process ID of the sender, and the sender can use an additional sequence field to distinguish between outstanding messages. However, message replies can be lost when kernel buffers are exhausted.

The *protocol* parameter specifies which messages an application listening on the routing socket is interested in seeing, based on the the address family of the `sockaddrs` present. Currently, you can specify `AF_INET` and `AF_INET6` to filter the messages seen by the listener, or alternatively, you can specify `AF_UNSPEC` to indicate that the listener is interested in all routing messages.

The kernel might reject certain messages, and indicates this by filling in the `rtm_errno` field of the `rt_msghdr` struct (see below). The following codes are returned:

`EEXIST` If requested to duplicate an existing entry

`ESRCH` If requested to delete a non-existent entry

`ENOBUFS` If insufficient resources were available to install a new route.

`EPERM` If the calling process does not have appropriate privileges to alter the routing table.

In the current implementation, all routing processes run locally, and the values for `rtm_errno` are available through the normal `errno` mechanism, even if the routing reply message is lost.

A process can avoid the expense of reading replies to its own messages by issuing a `setsockopt(3SOCKET)` call indicating that the `SO_USELOOPBACK` option at the `SOL_SOCKET` level is to be turned off. A process can ignore all messages from the routing socket by doing a `shutdown(3SOCKET)` system call for further input.

By default, underlying IP interfaces in an IPMP group are not visible to routing sockets. As such, routing sockets do not receive events related to underlying IP interface in an IPMP group. For consistency, when an IP interface is placed into an IPMP group, `RTM_DELADDR` messages are generated for each `IFF_UP` address that is not migrated to the corresponding IPMP IP interface and an `RTM_IFINFO` message is sent indicating the interface is down. Similarly, when an underlying interface is removed from an IPMP group, an `RTM_IFINFO` message is sent indicating the interface is again up and `RTM_NEWADDR` messages are generated for each `IFF_UP` address found on the interface.

The RT_AWARE socket option at the SOL_ROUTE level allows an application to indicate its awareness of certain features, which control routing socket behavior. The supported values are:

RTAW_DEFAULT — Default awareness.

RTAW_UNDER_IPMP — IPMP underlying interface awareness. When enabled, underlying IP interfaces in an IPMP group remain visible to the routing socket and events related to them continue to be generated.

An RTM_ADD request tied to an underlying IP interface in an IPMP group is translated to an RTM_ADD request for its corresponding IPMP IP interface. All routing socket requests other than RTM_ADD and RTM_GET fail when issued on an underlying IP interface in an IPMP group.

If a route is in use when it is deleted, the routing entry is marked down and removed from the routing table, but the resources associated with it are not reclaimed until all references to it are released.

The RTM_IFINFO, RTM_NEWADDR, and RTM_ADD messages associated with interface configuration (setting the IFF_UP bit) are normally delayed until after Duplicate Address Detection completes. Thus, applications that configure interfaces and wish to wait until the interface is ready can wait until RTM_IFINFO is returned and SIOCGLIFFLAGS shows that IFF_DUPLICATE is not set.

Messages

User processes can obtain information about the routing entry to a specific destination by using a RTM_GET message.

Messages include:

```
#define RTM_ADD      0x1   /* Add Route */
#define RTM_DELETE   0x2   /* Delete Route */
#define RTM_CHANGE   0x3   /* Change Metrics, Flags, or Gateway */
#define RTM_GET      0x4   /* Report Information */
#define RTM_LOSING   0x5   /* Kernel Suspects Partitioning */
#define RTM_REDIRECT 0x6   /* Told to use different route */
#define RTM_MISS     0x7   /* Lookup failed on this address */
#define RTM_LOCK     0x8   /* fix specified metrics */
#define RTM_OLDADD   0x9   /* caused by SIOCADDRT */
#define RTM_OLDDEL   0xa   /* caused by SIOCDELRT */
#define RTM_RESOLVE  0xb   /* request to resolve dst to LL addr */
#define RTM_NEWADDR  0xc   /* address being added to iface */
#define RTM_DELADDR  0xd   /* address being removed from iface */
#define RTM_IFINFO   0xe   /* iface going up/down etc. */
```

A message header consists of:

```
struct rt_msghdr {
  ushort_t rtm_msglen;    /* to skip over non-understood messages */
```

```
  uchar_t  rtm_version;   /* future binary compatibility */
  uchar_t  rtm_type;      /* message type */
  ushort_t rtm_index;     /* index for associated ifp */
  pid_t   rtm_pid;        /* identify sender */
  int     rtm_addrs;      /* bitmask identifying sockaddrs in msg */
  int     rtm_seq;        /* for sender to identify action */
  int     rtm_errno;      /* why failed */
  int     rtm_flags;      /*  flags,  incl  kern  &  message, e.g., DONE */
  int     rtm_use;        /* from rtentry */
  uint_t  rtm_inits;      /* which values we are initializing */

struct  rt_metrics rtm_rmx;   /* metrics themselves */
     };
```

where

```
struct rt_metrics {
  uint32_t rmx_locks;      /* Kernel must leave  these  values alone */
  uint32_t rmx_mtu;        /* MTU for this path */
  uint32_t rmx_hopcount;   /* max hops expected */
  uint32_t rmx_expire;     /* lifetime for route, e.g., redirect */
  uint32_t rmx_recvpipe;   /* inbound delay-bandwidth  product */
  uint32_t rmx_sendpipe;   /* outbound delay-bandwidth product */
  uint32_t rmx_ssthresh;   /* outbound gateway buffer limit */
  uint32_t rmx_rtt;        /* estimated round trip time */
  uint32_t rmx_rttvar;     /* estimated rtt variance */
  uint32_t rmx_pksent;     /* packets sent using this route */
 };

/* Flags include the values */


#define RTF_UP         0x1     /* route usable */
#define RTF_GATEWAY    0x2     /* destination is a gateway */
#define RTF_HOST       0x4     /* host entry (net otherwise) */
#define RTF_REJECT     0x8     /* host or net unreachable */
#define RTF_DYNAMIC    0x10    /* created dynamically(by redirect) */
#define RTF_MODIFIED   0x20    /* modified dynamically(by redirect) */
#define RTF_DONE       0x40    /* message confirmed */
#define RTF_MASK       0x80    /* subnet mask present */
#define RTF_CLONING    0x100   /* generate new routes on use */
#define RTF_XRESOLVE   0x200   /* external daemon resolves name */
#define RTF_LLINFO     0x400   /* generated by ARP */
#define RTF_STATIC     0x800   /* manually added */
#define RTF_BLACKHOLE  0x1000  /* just discard pkts (during updates) */
#define RTF_PRIVATE    0x2000  /* do not advertise this route */
#define RTF_PROTO2     0x4000  /* protocol specific routing flag #2 */
#define RTF_PROTO1     0x8000  /* protocol specific routing flag #1 */
```

```
#define RTF_MULTIRT    0x10000 /* multiroute */
#define RTF_SETSRC     0x20000 /* set default outgoing src address */
#define RTF_INDIRECT   0x40000 /* gateway not directly reachable */
#define RTF_KERNEL     0x80000 /* created by kernel; can't delete */

/* Specifiers for metric values in rmx_locks and rtm_inits are */

#define RTV_MTU        0x1     /* init or lock _mtu */
#define RTV_HOPCOUNT   0x2     /* init or lock _hopcount */
#define RTV_EXPIRE     0x4     /* init or lock _expire */
#define RTV_RPIPE      0x8     /* init or lock _recvpipe */
#define RTV_SPIPE      0x10    /* init or lock _sendpipe */
#define RTV_SSTHRESH   0x20    /* init or lock _ssthresh */
#define RTV_RTT        0x40    /* init or lock _rtt */
#define RTV_RTTVAR     0x80    /* init or lock _rttvar */

/* Specifiers for which addresses are present in  the  messages are */

#define RTA_DST        0x1     /* destination sockaddr present */
#define RTA_GATEWAY    0x2     /* gateway sockaddr present */
#define RTA_NETMASK    0x4     /* netmask sockaddr present */
#define RTA_GENMASK    0x8     /* cloning mask sockaddr present */
#define RTA_IFP        0x10    /* interface name sockaddr present */
#define RTA_IFA        0x20    /* interface addr sockaddr present */
#define RTA_AUTHOR     0x40    /* sockaddr for author of redirect */
#define RTA_BRD        0x80    /* for NEWADDR, broadcast or p-p dest addr */
```

See Also `ioctl(2)`, `setsockopt(3SOCKET)`, `shutdown(3SOCKET)`, `routing(7P)`

Notes Some of the metrics might not be implemented and return zero. The implemented metrics are set in `rtm_inits`.

The RTF_INDIRECT flag allows adding routes where the gateway is not directly reachable. When an indirect route is the best match for a packet to be sent or forwarded, then IP proceeds to lookup that gateway to find a route that is directly reachable. The RTF_INDIRECT flag can be used even if the gateway is directly reachable.

When the routing table contains several equal routes, that is, routes for the same destination and mask, then IP attempts to spread the traffic over those routes. The spreading is such that an individual transport connection uses the same route to avoid packet reordering as seen by e.g., TCP. The details of the spreading algoritm is not documented and is likely to evolve over time.

Name routing – system support for packet network routing

Description The network facilities provide general packet routing. The `routing` interface described here can be used to maintain the system's IPv4 routing table. It has been maintained for compatibility with older applications. The recommended interface for maintaining the system's routing tables is the routing socket, described at `route(7P)`. The routing socket can be used to manipulate both the IPv4 and IPv6 routing tables of the system. Routing table maintenance may be implemented in applications processes.

A simple set of data structures compose a "routing table" used in selecting the appropriate network interface when transmitting packets. This table contains a single entry for each route to a specific network or host. The routing table was designed to support routing for the Internet Protocol (IP), but its implementation is protocol independent and thus it may serve other protocols as well. User programs may manipulate this data base with the aid of two `ioctl(2)` commands, `SIOCADDRT` and `SIOCDELRT`. These commands allow the addition and deletion of a single routing table entry, respectively. Routing table manipulations may only be carried out by privileged user.

A routing table entry has the following form, as defined in `/usr/include/net/route.h`:

```
struct rtentry {
        unit_t   rt_hash;               /* to speed lookups */
        struct  sockaddr rt_dst;        /* key */
        struct  sockaddr rt_gateway;    /* value */
        short   rt_flags;               /* up/down?, host/net */
        short   rt_refcnt;              /* # held references */
        unit_t   rt_use;                /* raw # packets forwarded */
/*
 * The kernel does not use this field, and without it the structure is
 * datamodel independent.
 */
#if !defined(_KERNEL)
        struct  ifnet *rt_ifp;          /* the answer: interface to use */
#endif                                  /* !defined(_KERNEL) */
};
```

with *rt_flags* defined from:

```
#define RTF_UP 0x1         /* route usable */
#define RTF_GATEWAY 0x2    /* destination is a gateway */
#define RTF_HOST 0x4       /* host entry (net otherwise) */
```

There are three types of routing table entries: those for a specific host, those for all hosts on a specific network, and those for any destination not matched by entries of the first two types, called a wildcard route. Each network interface installs a routing table entry when it is initialized. Normally the interface specifies if the route through it is a "direct" connection to the destination host or network. If the route is direct, the transport layer of a protocol family

usually requests the packet be sent to the same host specified in the packet. Otherwise, the interface may be requested to address the packet to an entity different from the eventual recipient; essentially, the packet is forwarded.

Routing table entries installed by a user process may not specify the hash, reference count, use, or interface fields; these are filled in by the routing routines. If a route is in use when it is deleted, meaning its `rt_refcnt` is non-zero, the resources associated with it will not be reclaimed until all references to it are removed.

User processes read the routing tables through the `/dev/ip` device.

The *rt_use* field contains the number of packets sent along the route. This value is used to select among multiple routes to the same destination. When multiple routes to the same destination exist, the least used route is selected.

A wildcard routing entry is specified with a `zero` destination address value. Wildcard routes are used only when the system fails to find a route to the destination host and network. The combination of wildcard routes and routing redirects can provide an economical mechanism for routing traffic.

Errors

`EEXIST`	A request was made to duplicate an existing entry.
`ESRCH`	A request was made to delete a non-existent entry.
`ENOBUFS`	Insufficient resources were available to install a new route.
`ENOMEM`	Insufficient resources were available to install a new route.
`ENETUNREACH`	The gateway is not directly reachable. For example, it does not match the destination/subnet on any of the network interfaces.

Files

`/dev/ip`	IP device driver

See Also `route(1M)`, `ioctl(2)`, `route(7P)`

Name rtls – driver for Realtek 8139 fast Ethernet controllers

Description The `rtls` driver supports network interfaces based on the Realtek 8139 family of fast Ethernet controllers. These devices have an integrated 10BASE-T and 100BASE-TX PHY, and support IEEE 802.3 auto-negotiation of link speed and duplex mode.

The link settings can be viewed or modified using `dladm(1M)`.

Files

`/kernel/drv/rtls`	32–bit driver binary (x86)
`/kernel/drv/amd64/rtls`	64–bit driver binary (x86)
`/kernel/drv/sparcv9/rtls`	64–bit driver binary (SPARC)

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86

See Also `dladm(1M)`, `netstat(1M)`, `attributes(5)`, `ieee802.3(5)`, `dlpi(7P)`

Name rtw – RealTek 8180L 802.11b Wireless NIC driver

Description The `rtw` *802.11b* wireless NIC driver is a multi-threaded, loadable, clonable, GLDv3-based STREAMS driver supporting RealTek 8180L chipset-based NIC's.

Configuration The `rtw` driver performs auto-negotiation to determine the data rate and mode. Supported *802.11b* data rates are 1, 2, 5.5 and 11 Mbits/sec. The default is 11.

The `rtw` driver supports only BSS networks (also known as "ap" or "infrastructure" networks) and "open"(or "open-system") or "shared system" authentication.

Files `/dev/rtw*`
Special character device.

/kernel/drv/rtw
32-bit ELF kernel module (x86).

`/kernel/drv/amd64/rtw`
64-bit ELF kernel module (x86).

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWrtw
Interface stability	Committed

See Also `dladm(1M)`, `wificonfig(1M)`, `attributes(5)`, `gld(7D)`, `dlpi(7P)`

802.11 - Wireless LAN Media Access Control and Physical Layer Specification — IEEE, 2001

Name rum – Ralink RT2501/RT2601/RT73USB 802.11b/g Wireless Driver

Description The `rum` *802.11b/g* wireless NIC driver is a multi-threaded, loadable, clonable, GLDv3-based STREAMS driver supporting the Ralink RT2501/RT2601/RT73USB chipset-based NIC's.

Configuration The `rum` driver performs auto-negotiation to determine the data rate and mode. Supported 802.11b data rates are 1, 2, 5.5 and 11 Mbits/sec. Supported 802.11g data rates are 1, 2, 5.5, 11, 6, 9, 12, 18, 24, 36, 48 and 54 Mbits/sec. The `rum` driver supports only BSS networks (also known as "ap" or "infrastructure" networks) and "open" (or "open-system") or "shared system" authentication.

Files

`/dev/rum*`
Special character device.

`/kernel/drv/rum`
32-bit ELF kernel module. (x86)

`/kernel/drv/amd64/rum`
64-bit ELF kernel module. (x86)

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWrum
Interface Stability	Committed

See Also `dladm(1M)`, `wificonfig(1M)`, `attributes(5)`, `gld(7D)`, `dlpi(7P)`

802.11 - Wireless LAN Media Access Control and Physical Layer Specification - IEEE, 2001

Name rwd – Ralink RT2561/RT2561S/RT2661 IEEE802.11b/g wireless network driver

Description The rwd IEEE802.11b/g wireless network driver is a multithreaded, loadable, clonable, GLDv3-based STREAMS driver supporting Ralink RT2561/RT2561S/RT2661 IEEE802.11b/g wireless network driver.

Configuration The `rwd` driver performs auto-negotiation to determine the data rate and mode. The driver supports only BSS networks (also known as "ap" or "infrastructure" networks) and "open"(or "open-system") or "shared system" authentication. For wireless security, WEP encryption, WPA-PSk, and WPA2-PSK are currently supported. You can perform configuration and administration tasks using the `dladm`(1M)and `wificonfig`(1M) utilities.

Files

`/dev/rwd`	Special character device
`/kernel/drv/rwd`	32-bit ELF kernel module (x86)
`/kernel/drv/amd64/rwd`	64-bit ELF kernel module (x86)

Attributes See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWrwd
Interface Stability	Committed

See Also `dladm`(1M), `wificonfig`(1M), `attributes`(5), `gld`(7D), `dlpi`(7P)

IEEE802.11b/g - Wireless LAN Standard - IEEE, 2003

Name rwn – Ralink RT2700/2800 IEEE802.11 a/b/g/n wireless network device

Description The `rwn` IEEE802.11 a/b/g/n wireless driver is a multi-threaded, loadable, clonable, GLDv3-based STREAMS driver supporting Ralink RT2700/RT2800 IEEE802.11 a/b/g/n wireless network device.

Configuration The `rwn` driver performs auto-negotiation to determine the data rate and mode. The driver supports only BSS networks (also known as `ap` or `infrastructure` networks) and `open` (`open-system`) or `shared system` authentication.

For wireless security, WEP encryption, WPA-PSK, and WPA2-PSK are currently supported. You can perform configuration and administration tasks using the `dladm`(1M) and `wificonfig`(1M) utilities.

Files

`/dev/rwn`	Special character device
`/kernel/drv/rwn`	32-bit ELF kernel module, x86
`/kernel/drv/amd64/rwn`	64-bit ELF kernel module, x86

Attributes See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWrwn
Interface Stability	Committed

See Also `dladm`(1M), `wificonfig`(1M), `attributes`(5)

IEEE802.11b/g - Wireless LAN Standard - IEEE, 2003

Name sad – STREAMS Administrative Driver

Synopsis

```
#include <sys/types.h>

#include <sys/conf.h>

#include <sys/sad.h>

#include <sys/stropts.h>

int ioctl(int fildes, int command, int arg);
```

Description The STREAMS Administrative Driver provides an interface for applications to perform administrative operations on STREAMS modules and drivers. The interface is provided through `ioctl(2)` commands. Privileged operations may access the `sad` driver using `/dev/sad/admin`. Unprivileged operations may access the `sad` driver using `/dev/sad/user`.

The *fildes* argument is an open file descriptor that refers to the `sad` driver. The `command` argument determines the control function to be performed as described below. The *arg* argument represents additional information that is needed by this command. The type of *arg* depends upon the command, but it is generally an integer or a pointer to a `command`-specific data structure.

Command Functions The autopush facility (see `autopush(1M)`) allows one to configure a list of modules to be automatically pushed on a stream when a driver is first opened. Autopush is controlled by the following commands:

`SAD_SAP` Allows the administrator to configure the given device's autopush information. *arg* points to a `strapush` structure, which contains the following members:

```
unit_t   ap_cmd;
major_t  sap_major;
minor_t  sap_minor;
minor_t  sap_lastminor;
unit_t   sap_npush;
unit_t   sap_list [MAXAPUSH] [FMNAMESZ + 1];
```

The `sap_cmd` field indicates the type of configuration being done. It may take on one of the following values:

`SAP_ONE` Configure one minor device of a driver.

`SAP_RANGE` Configure a range of minor devices of a driver.

`SAP_ALL` Configure all minor devices of a driver.

`SAP_CLEAR` Undo configuration information for a driver.

The `sap_major` field is the major device number of the device to be configured. The `sap_minor` field is the minor device number of the device to be configured. The `sap_lastminor` field is used only with the `SAP_RANGE` command, which configures a range of minor devices between `sap_minor` and `sap_lastminor`,

inclusive. The minor fields have no meaning for the SAP_ALL command. The sap_npush field indicates the number of modules to be automatically pushed when the device is opened. It must be less than or equal to MAXAPUSH , defined in sad.h. It must also be less than or equal to NSTRPUSH, the maximum number of modules that can be pushed on a stream, defined in the kernel master file. The field sap_list is an array of NULL-terminated module names to be pushed in the order in which they appear in the list.

When using the SAP_CLEAR command, the user sets only sap_major and sap_minor. This will undo the configuration information for any of the other commands. If a previous entry was configured as SAP_ALL, sap_minor should be set to zero. If a previous entry was configured as SAP_RANGE , sap_minor should be set to the lowest minor device number in the range configured.

On failure, errno is set to the following value:

EFAULT *arg* points outside the allocated address space.

EINVAL The major device number is invalid, the number of modules is invalid, or the list of module names is invalid.

ENOSTR The major device number does not represent a STREAMS driver.

EEXIST The major-minor device pair is already configured.

ERANGE The command is SAP_RANGE and sap_lastminor is not greater than sap_minor, or the command is SAP_CLEAR and sap_minor is not equal to the first minor in the range.

ENODEV The command is SAP_CLEAR and the device is not configured for autopush.

ENOSR An internal autopush data structure cannot be allocated.

SAD_GAP Allows any user to query the sad driver to get the autopush configuration information for a given device. *arg* points to a strapush structure as described in the previous command.

The user should set the sap_major and sap_minor fields of the strapush structure to the major and minor device numbers, respectively, of the device in question. On return, the strapush structure will be filled in with the entire information used to configure the device. Unused entries in the module list will be zero-filled.

On failure, errno is set to one of the following values:

EFAULT *arg* points outside the allocated address space.

EINVAL The major device number is invalid.

ENOSTR The major device number does not represent a STREAMS driver.

ENODEV The device is not configured for autopush.

SAD_VML Allows any user to validate a list of modules (that is, to see if they are installed on the system). *arg* is a pointer to a `str_list` structure with the following members:

```
int     sl_nmods;
 struct  str_mlist    *sl_modlist;
```

The `str_mlist` structure has the following member:

```
char  l_name[FMNAMESZ+1];
```

`sl_nmods` indicates the number of entries the user has allocated in the array and `sl_modlist` points to the array of module names. The return value is 0 if the list is valid, 1 if the list contains an invalid module name, or −1 on failure. On failure, `errno` is set to one of the following values:

EFAULT *arg* points outside the allocated address space.

EINVAL The `sl_nmods` field of the `str_list` structure is less than or equal to zero.

See Also `Intro(2)`, `ioctl(2)`, `open(2)`

STREAMS Programming Guide

Diagnostics Unless otherwise specified, the return value from `ioctl()` is `0` upon success and −1 upon failure with `errno` set as indicated.

Name sata – Solaris SATA framework

Description Serial ATA is an interconnect technology designed to replace parallel ATA technology. It is used to connect hard drives, optical drives, removable magnetic media devices and other peripherals to the host system. For complete information on Serial ATA technology, visit the Serial ATA web site at http://*www.serialata.org*.

Up to 32 SATA devices may be plugged directly to each SATA HBA and up to 15 SATA devices may be plugged directly to each SATA port multiplier supported by the Solaris SATA framework. The actual number of pluggable devices my be lower, and is limited by the number of device ports on the SATA HBA or the SATA port multiplier. The maximum data rate is either 1.5Gb/sec. or 3.0Gb/sec., depending on the capability of a SATA device, port multiplier and SATA HBA controller.

The Solaris SATA framework adheres to the *Serial ATA 1.0a* specification and supports SATA-2 signaling speed 3.0Gb/sec. SATA devices that are connected to SATA HBAs controlled by a SATA framework-compliant HBA driver are treated by the system as SCSI devices. The Solaris SCSI disk driver (`sd(7D)`) is attached as a target driver for each device node created by the SATA framework. You can use the `cfgadm(1M)` utility to manage hot plugged and unplugged SATA devices.

Files

`/kernel/misc/sata`	32–bit ELF kernel module (x86).
`/kernel/misc/amd64/sata`	64–bit ELF kernel module (x86).

Attributes See `attributes(5)` for descriptions of the following attribute:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWckr

See Also `cfgadm(1M)`, `prtconf(1M)`, `cfgadm_sata(1M)`, `attributes(5)`, `ahci(7D)`, `marvell88sx(7D)`, `nv_sata(7D)`, `sd(7D)`, `si3124(7D)`

Serial ATA 1.0a Specification — Serial ATA International Organization.

Serial ATA II (Extension to Serial ATA 1.0.a.) — Serial ATA International Organization.

http://www.sun.com/

Diagnostics The messages described below may appear on the system console as well as being logged. All messages are presented in one of the following formats and are followed by the diagnostic message:

```
sata: WARNING: <controller/devices/.. path>:
```

or:

```
sata: NOTICE: <controller/devices/.. path>:
```

...where <controller/devices/.. path> identifies a specific SATA HBA issuing a diagnostic message shown below.

SATA port X: link lost.
: Communication (via serial link) between the HBA and the device plugged to the specified SATA device port has been lost.

SATA port X: link established.
: Communication (via serial link) between the HBA and the device plugged to the specified SATA device port has been established.

SATA port X: device reset.
: The device plugged to the specified SATA device port has been reset. The reset may be due to a communication or command error, command timeout, or an explicit request from the host.

SATA port X failed.
: The specified SATA device port failed and is in an unusable state. You can change the port state by deactivating the port and activating it again using cfgadm SATA hardware-specific commands (see cfgadm_sata(1M)).

SATA port X error.
: An error was detected in specified SATA device port operations.

SATA device detached at port X.
: Communication (via serial link) between the HBA and the device plugged to the specified SATA device port has been lost and could not be re-established. The SATA framework assumes that the device is unplugged from the specified SATA device port.

SATA device detected at port X.
: Communication(via serial link) between the HBA and the device plugged to the specified empty SATA device port has been established. The SATA framework assumes that the new device is plugged to the specified SATA device port.

SATA disk device at port X.
: This message is followed by a disk description specifying the disk vendor, serial number, firmware revision number and the disk capabilities.

SATA CD/DVD (ATAPI) device at port X.
: This message is followed by a SATA CD/DVD description specifying the DVD vendor, serial number, firmware revision number and the DVD capabilities.

SATA device at port X cannot be configured. Application(s) accessing previously attached device have to release it before newly inserted device can be made accessible.
: The port cannot be configured because there is application using the previous attached device, so the application must release it, then the newly inserted device can be configured.

Application(s) accessing previously attached SATA device have to release it before newly inserted device can be made accessible.

The target node remained and it belongs to a previously attached device. This happens when the file was open or the node was waiting for resources at the time the associated device was removed. Instruct event daemon to retry the cleanup later.

sata: error recovery request for non-attached device at cport X.

When error recovery is requested, the device is not yet attached.

SATA device at port X is not power-managed.

When property `pm-capable` on the target device node setting fails, the SATA device won't be power-managed.

SATA disk device at port X does not support LBA.

The disk device plugged into specified SATA device port does not support LBA addressing and cannot be used.

Cannot identify SATA device at port X - device is attached.

IDENTIFY (PACKET) DEVICE data cannot be retrieved successfully after the device is attached to the SATA port.

sata: <HBA driver name><instance number>:hba attached failed.

The SATA HBA instance attach operation failed. This HBA instance cannot be configured and is not available.

sata: invalid ATAPI cdb length<command cdb length>.

The length of the command cdb is greater than that the device can support.

sata: invalid sata_hba_tran version X for driver <HBA driver name>.

The specified SATA HBA driver and the SATA framework are incompatible. The driver cannot attach and SATA HBAs controlled by this driver (and devices plugged to this SATA HBA ports) are not available.

sata_hba_attach: cannot create SATA attachment point for port X.

The specified SATA device port cannot be configured in the system and a device plugged to this port could not be not be configured and used.

sata_create_target_node: cannot create target node for device at port X.

The device target node for the device plugged to the specified SATA device port could not be created. As a result, the device cannot be configured and used.

Name scfd – System Control Facility (SCF) driver

Synopsis

```
scfd@unit-address
```

Description The System Control Facility (SCF) driver is a device driver that communicates with the eXtended System Control Facility (XSCF) firmware on a SPARC Enterprise Server.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcakr
Interface Stability	Private

See Also `attributes(5)`

Name schpc – StarCat Hot Plug Controller Driver

Description The `schpc` driver controls all hot-plug operations on high-end Sun Fire E15K and E25K enterprise servers.

Files `/platform/SUNW,Sun-Fire-15000/kernel/drv/sparcv9/schpc`
64-bit ELF kernel module.

`/platform/SUNW,Sun-Fire-15000/kernel/drv/schpc.conf`
Driver configuration file.

See Also `cfgadm(1M)`

Name scsa1394 – SCSI to 1394 bridge driver

Synopsis `unit@GUID`

Description The `scsa1394` driver is a 1394 target and an SCSA HBA driver that supports 1394 mass storage devices compliant with the *Serial Bus Protocol 2 (SBP-2)* specification. It supports both bus-powered and self-powered 1394 mass storage devices.

The `scsa1394` nexus driver maps SCSA target driver requests to SBP-2 Operation Request Blocks (ORB's).

The `scsa1394` driver creates a child device info node for each logical unit (LUN) on the mass storage device. The standard Solaris SCSI disk driver is attached to those nodes. Refer to `sd`(7D).

This driver supports multiple LUN devices and creates a separate child device info node for each LUN. All child LUN nodes attach to `sd`(7D).

In previous releases, all 1394 mass storage devices were treated as removable media devices and managed by `rmformat`(1) and volume management software. In the current release, however, only mass storage devices with a removable bit (RMB) value of `1` are removable. (The RMB is part of the device's SCSI INQUIRY data.) See SCSI specifications T10/995D Revision 11a, T10/1236-D Revision 20 or T10/1416-D Revision 23 for more information. However, for backward compatibility, all 1394 mass storage devices can still be managed by `rmformat`(1). With or without a volume manager, you can mount, eject, hot remove and hot insert a 1394 mass storage device as the following sections explain.

Using Volume Management Mass storage devices are managed by a volume manager. Software that manages removable media creates a device nickname that can be listed with `eject`(1) or `rmmount`(1). A device that is not mounted automatically can be mounted using `rmmount`(1) under `/rmdisk/`*label*. Note that the `mount`(1M) and `mount`(1M) commands do not accept nicknames; you must use explicit device names with these commands.

See `rmmount`(1) to unmount the device and `eject`(1) to eject the media. If the device is ejected while it is mounted, volume management software unmounts the device before ejecting it. It also might kill any active applications that are accessing the device.

Volume management software is hotplug-aware and normally mounts file systems on USB mass storage devices if the file system is recognized. Before hot removing the USB device, use `eject`(1) to unmount the file system.

You can disable the automatic mounting and unmounting of removable devices by inserting a entry for a removable device in `/etc/vfstab`. In this entry, you must set the `mount at boot` field to `no`. See `vfstab`(4). See the *System Administration Guide, Volume I* and *Solaris Common Desktop Environment: User's Guide* for details on how to manage a removable device with CDE and Removable Media Manager. See `dtfile.1X` under CDE for information on how to use Removable Media Manager.

Using `mount` And `umount` Use `mount(1M)` to explicitly mount the device and `umount(1M)` to unmount the device. Use `eject(1)` to eject the media. After you have explicitly mounted a removable device, you cannot use a nickname as an argument to `eject`.

Removing the storage device while it is being accessed or mounted fails with a console warning. To hot remove the storage device from the system, unmount the file system, then kill all applications accessing the device. Next, hot remove the device. A storage device can be hot inserted at any time.

For a comprehensive listing of (non-bootable) 1394 mass-storage devices that are compatible with this driver, see `www.sun.com/io`.

Device Special Files Block special file names are located in `/dev/dsk`. Raw file names are located in `/dev/rdsk`. Input/output requests to the devices must follow the same restrictions as those for SCSI disks. Refer to `sd(7D)`.

ioctls Refer to `cdio(7I)` and `dkio(7I)`.

Errors Refer to `sd(7D)`.

Files The device special files for the 1394 mass storage device are created like those for a SCSI disk. Refer to `sd(7D)`.

`/dev/dsk/c`*n*`t`*n*`d`*n*`s`*n*	Block files
`/dev/rdsk/c`*n*`t`*n*`d`*n*`s`*n*	Raw files
`/vol/dev/aliases/rmdisk0`	Symbolic link to the character device for the media in removable drive 0. This is a generic removable media device.
`/kernel/drv/scsa1394`	32–bit x86 ELF kernel module
`/kernel/drv/amd64/scsa1394`	64–bit x86 ELF kernel module
`/kernel/drv/sparcv9/scsa1394`	64–bit SPARC ELF kernel module

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86, PCI-based systems
Availability	SUNWscsa1394

See Also `cdrw(1)`, `eject(1)`, `rmformat(1)`, `rmmount(1)`, `cfgadm_scsi(1M)`, `fdisk(1M)`, `mount(1M)`, `umount(1M)`, `dtfile.1X` (in CDE man pages), `scsi(4)`, `vfstab(4)`, `attributes(5)`, `hci1394(7D)`, `sd(7D)`, `pcfs(7FS)`, `cdio(7I)`, `dkio(7I)`

IEEE Std 1394-1995 Standard for a High Performance Serial Bus

ANSI NCITS 325-1998 - Serial Bus Protocol 2 (SBP-2)

System Administration Guide: Devices and File Systems

SCSI Specification *T10/995D Revision 11a* — March 1997

SCSI Specification*T10/1236-D Revision 20* — July 2001

SCSI Specification*T10/1416-D Revision 23*— May 2005

Solaris Common Desktop Environment: User's Guide

http://www.sun.com/io

Name scsa2usb – SCSI to USB bridge driver

Synopsis `storage@unit-address`

Description The `scsa2usb` driver is a USBA (Solaris USB architecture) compliant nexus driver that supports the *USB Mass Storage Bulk Only Transport Specification 1.0* and *USB Control/Bulk/Interrupt (CBI) Transport Specification 1.0*. The `scsa2usb` driver also supports USB storage devices that implement CBI Transport without the interrupt completion for status (that is, Control/Bulk (CB) devices.) It supports bus-powered and self-powered USB mass storage devices. This nexus driver is both a USB client driver and a SCSA HBA driver. As such, the `scsa2usb` driver only supports storage devices that utilize the above two transports.

The `scsa2usb` driver also supports a `ugen(7D)` interface allowing raw access to the device, for example by `libusb(3LIB)` applications, bypassing the child `sd(7D)` or `st(7D)` driver. Because a libusb application might change the state of the device, you should not access the disk or tape concurrently.

The `scsa2usb` nexus driver maps SCSA target driver requests to USBA client driver requests.

The `scsa2usb` driver creates a child device info node for each logical unit (LUN) on the mass storage device. The standard Solaris SCSI disk driver or tape driver is attached to those nodes. Refer to `sd(7D)` or `st(7D)`.

This driver supports multiple LUN devices and creates a separate child device info node for each LUN. All child LUN nodes attach to `sd(7D)` for disks or `st(7D)` for tapes.

In previous releases, all USB disk storage devices were treated as removable media devices and managed by `rmformat(1)` and volume management software. In the current release, however, only disk storage devices with a removable bit (RMB) value of `1` are removable. (The RMB is part of the device's SCSI INQUIRY data.) See SCSI specifications T10/995D Revision 11a, T10/1236-D Revision 20 or T10/1416-D Revision 23 for more information. However, for backward compatibility, all USB disk storage devices can still be managed by `rmformat(1)`. With or without a volume manager, you can mount, eject, hot remove and hot insert a 1394 mass storage device as the following sections explain.

Some devices may be supported by the USB mass storage driver even though they do not identify themselves as compliant with the USB mass storage class.

The `scsa2usb.conf` file contains an `attribute-override-list` that lists the vendor ID, product ID, and revision for matching mass storage devices, as well as fields for overriding the default device attributes. The entries in this list are commented out by default and may be uncommented to enable support of particular devices.

Follow the information given in the `scsa2usb.conf` file to see if a particular device can be supported using the override information. Also see http://*www.sun.com/io*. For example, by adding the following to the `scsa2usb.conf` file, many USB memory sticks and card readers might operate more reliably:

```
attribute-override-list = "vid=* reduced-cmd-support=true";
```

Note that this override applies to all USB mass storage devices and might be inappropriate for a USB CD writer. If so, you can add an entry for each device to the attribute override list.

If USB mass storage support is considered a security risk, this driver can be disabled in `/etc/system` as follows:

```
exclude: scsa2usb
```

Alternatively, you can disable automatic handling of a device as described in the following subsection.

Using Volume Management

Disk storage devices are managed by Volume Manager. Software that manages removable media creates a device nickname that can be listed with `eject(1)` or `rmmount(1)`. A device that is not mounted automatically can be mounted using `rmmount(1)` under `/rmdisk/`*label*. Note that the `mount(1M)` and `mount(1M)` commands do not accept nicknames; you must use explicit device names with these commands.

See `rmmount(1)` to unmount the device and `eject(1)` to eject the media. If the device is ejected while it is mounted, volume management software unmounts the device before ejecting it. It also might kill any active applications that are accessing the device.

Volume management software is hotplug-aware and normally mounts file systems on USB mass storage devices if the file system is recognized. Before hot removing the USB device, use `eject(1)` to unmount the file system. After the device is removed, a console warning, such as "The disconnected device was busy, please reconnect," might display. The warning is harmless and you can ignore it.

You can disable the automatic mounting and unmounting of removable devices by inserting a entry for a removable device in `/etc/vfstab`. In this entry, you must set the `mount at boot` field to `no`. See `vfstab(4)`. See the *System Administration Guide, Volume I* and *Solaris Common Desktop Environment: User's Guide* for details on how to manage a removable device with CDE and Removable Media Manager. See `dtfile.1X` under CDE for information on how to use Removable Media Manager.

Using `mount` and `umount`

Use `mount(1M)` to explicitly mount the device and `umount(1M)` to unmount the device. Use `eject(1)` to eject the media. After you have explicitly mounted a removable device, you cannot use a nickname as an argument to `eject`.

Removing the disk device while it is being accessed or mounted fails with a console warning. To hot remove the disk device from the system, unmount the file system, then kill all applications accessing the device. Next, hot remove the device. A storage device can be hot inserted at any time.

For a comprehensive listing of (non-bootable) USB mass-storage devices that are compatible with this driver, see `www.sun.com/io`.

Device Special Files Disk block special file names are located in `/dev/dsk`, while raw file names are located in `/dev/rdsk`. Tape raw file names are located in `/dev/rmt`. Input/output requests to the devices must follow the same restrictions as those for SCSI disks or tapes. Refer to `sd`(7D) or `st`(7D).

ioctls Refer to `dkio`(7I) and `cdio`(7I).

Errors Refer to `sd`(7D) for disks or `st`(7D) for tapes.

Files The device special files for the USB mass storage device are created like those for a SCSI disk or SCSI tape. Refer to `sd`(7D) or `st`(7D).

`/dev/dsk/c`*n*`t`*n*`d`*n*`s`*n*	Block files for disks.
`/dev/rdsk/c`*n*`t`*n*`d`*n*`s`*n*	Raw files for disks.
/dev/usb/*/*/*	ugen(7D) nodes
`/dev/rmt/[0- 127][l,m,h,u,c][b][n]`	Raw files for tapes.
`/vol/dev/aliases/zip0`	Symbolic link to the character device for the media in Zip drive 0
`/vol/dev/aliases/jaz0`	Symbolic link to the character device for the media in Jaz drive 0.
`/vol/dev/aliases/rmdisk0`	Symbolic link to the character device for the media in removable drive 0. This is a generic removable media device.
`/kernel/drv/scsa2usb`	32–bit x86 ELF kernel module
`/kernel/drv/amd64/scsa2usb`	64–bit x86 ELF kernel module
`/kernel/drv/sparcv9/scsa2usb`	64–bit SPARC ELF kernel module
`/kernel/drv/scsa2usb.conf`	Can be used to override specific characteristics.

Attributes See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86, PCI-based systems
Availability	SUNWusb

See Also `cdrw`(1), `eject`(1), `rmformat`(1), `rmmount`(1), `cfgadm_scsi`(1M), `cfgadm_usb`(1M), `fdisk`(1M), `mount`(1M), `umount`(1M), `dtfile.1X` (in CDE man pages), `libusb`(3LIB), `scsi`(4), `vfstab`(4), `attributes`(5), `ieee1394`(7D)`sd`(7D), `st`(7D), `ugen`(7D), `usba`(7D), `pcfs`(7FS), `cdio`(7I), `dkio`(7I)

Writing Device Drivers

System Administration Guide, Volume I

Solaris Common Desktop Environment: User's Guide

Universal Serial Bus Specification 2.0

Universal Serial Bus Mass Storage Class Specification Overview 1.0

Universal Serial Bus Mass Storage Class Bulk-Only Transport Specification 1.0

Universal Serial Bus Mass Storage Class Control/Bulk/Interrupt (CBI) Transport Specification 1.0

System Administration Guide: Basic Administration

SCSI Specification *T10/995D Revision 11a* — March 1997

SCSI Specification*T10/1236-D Revision 20* — July 2001

SCSI Specification*T10/1416-D Revision 23*— May 2005

http://*www.sun.com/io*

Diagnostics Refer to `sd`(7D) and `st`(7D).

In addition to being logged, the following messages may appear on the system console. All messages are formatted in the following manner:

```
Warning: <device path> (scsa2usb<instance number>): Error Message...
```

Cannot access <device>. Please reconnect.
: There was an error in accessing the mass-storage device during reconnect. Please reconnect the device.

Device is not identical to the previous one on this port. Please disconnect and reconnect.
: Another USB device has been inserted on a port that was connected to a mass-storage device. Please disconnect the USB device and reconnect the mass-storage device back into that port.

Reinserted device is accessible again.
: The mass-storage device that was hot-removed from its USB slot has been re-inserted to the same slot and is available for access.

Please disconnect and reconnect this device.
: A hotplug of the device is needed before it can be restored.

The following messages may be logged into the system log. They are formatted in the following manner:

```
<device path><scsa2usb<instance number>): message...
```

Invalid <record> in scsa2usb.conf file entry.
: An unrecognized record was specified in the `scsa2usb.conf` file.

Pkt submitted with 0 timeout which may cause indefinite hangs.
An application submitted a request but did not specify a timeout.

Syncing not supported.
Syncing after a panic is not supported. The filesystem may be corrupted.

scsa2usb.conf override: <record>.
An override record specified in `scsa2usb.conf` was applied. Examples of an override record applied to a device with vendor ID 123 and product ID 456 are:

```
vid=0x123 pid=0x456 reduced-cmd-support=true

    or

vid=* reduced-cmd-support=true
```

...meaning that the override record is applied to this device and all other USB mass storage devices.

Notes The Zip 100 drive does not comply with *Universal Serial Bus Specification 1.0* and cannot be power managed. Power Management support for Zip 100 has been disabled.

If the system panics while a UFS file system is mounted on the mass storage media, no syncing will take place for the disk mass-storage device. (Syncing is not supported by the `scsa2usb` driver.) As a result, the file system on the media will not be consistent on reboot.

If a PCFS file system is mounted, no syncing is needed and the filesystem will be consistent on reboot.

If a mass-storage device is busy, system suspend cannot proceed and the system will immediately resume again.

Attempts to remove a mass-storage device from the system will fail. The failure will be logged to the console. An attempt to replace the removed device with some other USB device will also fail. To successfully remove a USB mass-storage device you must "close" all references to it.

An Iomega Zip 100Mb disk cannot be formatted on an Iomega Zip250 drive. See the Iomega web site at *http://www.iomega.com* for details.

Concurrent I/O to devices with multiple LUNs on the same device is not supported.

Some USB CD-RW devices may perform inadequately at their advertised speeds. To compensate, use USB CD-RW devices at lower speeds (2X versus 4X). See `cdrw(1)` for details.

This driver also supports CBI devices that do not use USB interrupt pipe for status completion.

Name scsi_vhci – SCSI virtual host controller interconnect driver

Description The `scsi_vhci` driver is a SCSA compliant pseudo nexus driver that supports Solaris operating system I/O multipathing services for SCSI-3 devices. This driver introduces a fundamental restructuring of the Solaris device tree to enable a multipath device to be represented as single device instance rather than as an instance per physical path as in earlier Solaris versions.

The logical units (LUNs) associated multipath SCSI target devices managed by this driver are identified and represented by using the SCSI-3 VPD page (0x83) LUN global unique identifier (GUID) represented as hexadecimal number (64/128 bits)

Symbolic links in `/dev/[r]dsk` and `/dev/scsi/changer` continue to adhere to the c*N*t*N*d*N*s*N* format. c*N* is the logical controller number assigned to this driver instance. tN is the GUID.

Symbolic links in `/dev/rmt/#[l|m|h|c|u],[b],[n]` also adhere to the same format as non-multipath devices. Because of persistent binding of tape devices, you may want to remove old non-multipath links when enabling them for multipath.

The following is an example of a system with an A5000 storage array:

```
...
    /dev/rdsk/c4t200000203709C3F5d0s0 -> ../../devices/
         scsi_vhci/ssd@g200000203709c3f5:a,raw
    ...
    /dev/rdsk/c4t200000203709C3F5d0s7 -> ../../devices/
         scsi_vhci/ssd@g200000203709c3f5:h,ra
    ...
```

The following is an example of a system with a T300 storage array:

```
    ...
    /dev/rdsk/c1t60020F200000033939C2C2B60008D4AEd0s0 ->
         ../../devices/scsi_vhci/
     ssd@g60020f200000033939a2c2b60008d4ae:a,raw
    ...
    /dev/rdsk/c1t60020F200000033939A2C2B60008D4AEd0s7 ->
         ../../devices/scsi_vhci/
ssd@g60020f200000033939a2c2b60008d4ae:h,raw
```

The `scsi_vhci` driver receives naming and transport services from one or more physical HBA (host bus adapter) devices. To support multi-pathing, a physical HBA driver must have its multipathing enabled and comply with the multipathing services provided by this driver.

The `scsi_vhci` driver supports the standard functions provided by the SCSA interface.

Configuration For each candidate SCSI target device, the scsi_vhci code must identify a failover module to support the device. If a failover module can't be identified, the device will not function under `scsi_vhci` multipathing control. For SCSI target devices that support the standard Target Port Group Select, no special vendor/product knowledge is needed. For other SCSI target devices, each failover module understands which devices it supports.

When autoconfiguration does not result in the desired configuration, a vendor/product specific override mechanism is available. This `scsi_vhci.conf` base mechanism can be used to direct a device to a specific failover module (or to indicate that a device should not be under scsi_vhci multipathing control by way of `NONE`). In `scsi_vhci.conf`, the property `'scsi-vhci-failover-override'` defines overrides in `scsi_get_device_type_string(9F)` form. To add a third-party (non-Sun) symmetric storage device to run under scsi_vhci (and thereby take advantage of scsi_vhci multipathing), you add the vendor ID and product ID for the device, as those strings are returned by the SCSI Inquiry command. For example, to add a device from a vendor with the ID of "`Acme`" and a product ID of "`MSU`", you would add:

```
device-type-scsi-options-list =
   "Acme    MSU", "f_sym",
```

In addition to "`Acme`", you also might want to add another entry, for example, a device from "`XYZ`" vendor with a product ID of "`ABC`":

```
scsi-vhci-failover-override =
   "Acme    MSU", "f_sym",
   "XYZ     ABC", "f_sym";
```

As a last override, you might add an entry so that no devices from "`ABC`" vendor use `scsi_vhci` multipathing:

```
scsi-vhci-failover-override =
    "Acme    MSU",    "f_sym",
    "XYZ     ABC",    "f_sym",
    "ABC     ",       "NONE";
```

Files

`/kernel/drv/sparcv9/scsi_vhci`	64–bit kernel module (SPARC).
`/kernel/drv/scsi_vhci`	32-bit kernel module (x86).
`/kernel/drv/amd64/scsi_vhci`	64-bit kernel module (amd64).
`/kernel/drv/scsi_vhci.conf`	Driver configuration file.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	PCI-based systems
Availability	SUNWckr

See Also eeprom(1M), prtconf(1M), stmsboot(1M), mpathadm(1M), attributes(5), fcp(7D), fctl(7D), fp(7d), mpt(7D), ssd(7D), sd(7D), st(7D), sgen(7D), scsi_abort(9F), scsi_get_device_type_scsi_options(9F), scsi_get_device_type_string(9F), scsi_ifgetcap(9F), scsi_reset(9F), scsi_transport(9F), scsi_inquiry(9S), scsi_extended_sense(9S), scsi_pkt(9S)

Writing Device Drivers

Small Computer System Interface-3 (SCSI-3)

Notes In previous releases, the `scsi_vhci.conf` file supported the mpxio-disable property, which allowed you to disable Solaris I/O multipathing on a system-wide basis. This property is not present in the current release of the Solaris operating system. Multipathing is always enabled in scsi_vhci. If you want to disable multipathing, use the mechanisms provided by the HBA drivers. See fp(7d) and mpt(7D).

In previous releases, the override mechanism was based on the scsi_get_device_type_scsi_options(9F) defined "`device-type-scsi-options-list`" property. During upgrade, `scsi_vhci.conf` is converted to the new form. After upgrade, a `scsi_vhci.conf` modification based on the old mechanism is silently ignored.

In previous releases, Solaris I/O multipathing was also known as MPxIO and Sun StorEdge Traffic Manager (STMS).

Name sctp, SCTP – Stream Control Transmission Protocol

Synopsis

```
#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_STREAM, IPPROTO_SCTP);

s = socket(AF_INET, SOCK_SEQPACKET, IPPROTO_SCTP);

s = socket(AF_INET6, SOCK_STREAM, IPPROTO_SCTP);

s = socket(AF_INET6, SOCK_SEQPACKET, IPPROTO_SCTP);
```

Description SCTP is a transport protocol layered above the Internet Protocol (IP), or the Internet Protocol Version 6 (IPv6). SCTP provides a reliable, session oriented, flow-controlled, two-way transmission of data. It is a message– oriented protocol and supports framing of individual messages boundaries. An SCTP association is created between two endpoints for data transfer which is maintained during the lifetime of the transfer. An SCTP association is setup between two endpoints using a four-way handshake mechanism with the use of a cookie to guard against some types of denial of service (DoS) attacks. These endpoints may be represented by multiple IP addresses.

An SCTP message includes a common SCTP header followed by one or more chunks. Included in the common header is a 32-bit field which contains the checksum (computed using CRC-32c polynomial) of the entire SCTP packet.

SCTP transfers data payloads in the form of DATA chunks. Each DATA chunk contains a Transmission Sequence Number (TSN), which governs the transmission of messages and detection of loss. DATA chunk exchanges follow the Transmission Control Protocol's (TCP) Selective ACK (SACK) mechanism. The receiver acknowledges data by sending SACK chunks, which not only indicate the cumulative TSN range received, but also non-cumulative TSNs received, implying gaps in the received TSN sequence. SACKs are sent using the delayed acknowledgment method similar to TCP, that is, one SCTP per every other received packet with an upper bound on the delay (when there are gaps detected the frequency is increased to one every received packet). Flow and congestion control follow TCP algorithms: Slow Start, Congestion Avoidance, Fast Recovery and Fast retransmit. But unlike TCP, SCTP does not support half-close connection and “urgent” data.

SCTP is designed to support a number of functions that are critical for telephony signalling transport, including multi-streaming. SCTP allows data to be partitioned into multiple streams that have the property of independent sequenced delivery so that message loss in any one stream only affects delivery within that stream. In many applications (particularly telephony signalling), it is only necessary to maintain sequencing of messages that affect some resource. Other messages may be delivered without having to maintain overall sequence integrity. A DATA chunk on an SCTP association contains the Stream Id/Stream Sequence Number pair, in addition to the TSN, which is used for sequenced delivery within a stream.

SCTP uses IP's host level addressing and adds its own per-host collection of port addresses. The endpoints of an SCTP association are identified by the combination of IP address(es) and an SCTP port number. By providing the ability for an endpoint to have multiple IP addresses, SCTP supports multi-homing, which makes an SCTP association more resilient in the presence of network failures (assuming the network is constructed to provided redundancy). For a multi-homed SCTP association, a single address is used as the primary address, which is used as the destination address for normal DATA chunk transfers. Retransmitted DATA chunks are sent over alternate address(es) to increase the probability of reaching the remote endpoint. Continued failure to send DATA chunks over the primary address results in selecting an alternate address as the primary address. Additionally, SCTP monitors the accessibility of all alternate addresses by sending periodic "heartbeats" chunks. An SCTP association supports multi-homing by exchanging the available list of addresses during association setup (as part of its four-way handshake mechanism). An SCTP endpoint is associated with a local address using the `bind(3SOCKET)` call. Subsequently, the endpoint can be associated with additional addresses using `sctp_bindx(3SOCKET)`. By using a special value of `INADDR_ANY` with IP or the unspecified address (all zeros) with IPv6 in the `bind()` or `sctp_bindx()` calls, an endpoint can be bound to all available IP or IPv6 addresses on the system.

SCTP uses a three-way mechanism to allow graceful shutdown, where each endpoint has confirmation of the DATA chunks received by the remote endpoint prior to completion of the shutdown. An Abort is provided for error cases when an immediate shutdown is needed.

Applications can access SCTP using the socket interface as a `SOCK_STREAM` (one-to-one style) or `SOCK_SEQPACKET` (one-to-many style) socket type.

One-to-one style socket interface supports similar semantics as sockets for connection oriented protocols, such as TCP. Thus, a passive socket is created by calling the `listen(3SOCKET)` function after binding the socket using `bind()`. Associations to this passive socket can be received using `accept(3SOCKET)` function. Active sockets use the `connect(3SOCKET)` function after binding to initiate an association. If an active socket is not explicitly bound, an implicit binding is performed. If an application wants to exchange data during the association setup phase, it should not call `connect()`, but use `sendto(3SOCKET)`/`sendmsg(3SOCKET)` to implicitly initiate an association. Once an association has been established, `read(2)` and `write(2)` can used to exchange data. Additionally, `send(3SOCKET)`, `recv(3SOCKET)`, `sendto()`, `recvfrom(3SOCKET)`, `sendmsg()`, and `recvmsg(3SOCKET)` can be used.

One-to-many socket interface supports similar semantics as sockets for connection less protocols, such as UDP (however, unlike UDP, it does not support broadcast or multicast communications). A passive socket is created using the `listen()` function after binding the socket using `bind()`. An `accept()` call is not needed to receive associations to this passive socket (in fact, an `accept()` on a one-to-many socket will fail). Associations are accepted automatically and notifications of new associations are delivered in `recvmsg()` provided notifications are enabled. Active sockets after binding (implicitly or explicitly) need not call

`connect()` to establish an association, implicit associations can be created using `sendmsg()/recvmsg()` or `sendto()/recvfrom()` calls. Such implicit associations cannot be created using `send()` and `recv()` calls. On an SCTP socket (one-to-one or one-to-many), an association may be established using `sendmsg()`. However, if an association already exists for the destination address specified in the *msg_name* member of the *msg* parameter, `sendmsg()` must include the association id in *msg_iov* member of the *msg* parameter (using `sctp_sndrcvinfo` structure) for a one-to-many SCTP socket. If the association id is not provided, `sendmsg()` fails with `EADDRINUSE`. On a one-to-one socket the destination information in the *msg* parameter is ignored for an established association.

A one-to-one style association can be created from a one-to-many association by branching it off using the `sctp_peeloff(3SOCKET)` call; `send()` and `recv()` can be used on such peeled off associations. Calling `close(2)` on a one-to-many socket will gracefully shutdown all the associations represented by that one-to-many socket.

The `sctp_sendmsg(3SOCKET)` and `sctp_recvmsg(3SOCKET)` functions can be used to access advanced features provided by SCTP.

SCTP provides the following socket options which are set using `setsockopt(3SOCKET)` and read using `getsockopt(3SOCKET)`. The option level is the protocol number for SCTP, available from `getprotobyname(3SOCKET)`.

`SCTP_NODELAY`
: Turn on/off any Nagle-like algorithm (similar to `TCP_NODELAY`).

`SO_RCVBUF`
: Set the receive buffer.

`SO_SNDBUF`
: Set the send buffer.

`SCTP_AUTOCLOSE`
: For one-to-many style socket, automatically close any association that has been idle for more than the specified number of seconds. A value of '0' indicates that no associations should be closed automatically.

`SCTP_EVENTS`
: Specify various notifications and ancillary data the user wants to receive.

`SCTP_STATUS`
: Retrieve current status information about an SCTP association.

In addition SCTP provides the following option to handle gathering of a limited set of per endpoint association statistics from a one-to-one socket.

`SCTP_GET_ASSOC_STATS` Gather and reset per endpoint association statistics.

Example Usage:

```
#include <netinet/sctp.h>

struct sctp_assoc_stats stat;
int rc;

int32_t len = sizeof (stat);

/*
 * Per endpoint stats use the socket descriptor for sctp association.
 */

/* Gather per endpoint association statistics */
rc = getsockopt(sd, IPPROTO_SCTP, SCTP_GET_ASSOC_STATS, &stat, &len);

-----
sctp.h

 /*
  * SCTP socket option used to read per endpoint association statistics.
  */
  #define SCTP_GET_ASSOC_STATS          24

 /*
  * A socket user request reads local per endpoint association stats.
  * All stats are counts except sas_maxrto, which is the max value
  * since the last user request for stats on this endpoint.
  */
  typedef struct sctp_assoc_stats {
      uint64_t  sas_rtxchunks;   /* Retransmitted Chunks */
      uint64_t  sas_gapcnt;      /* Gap Acknowledgements Received */
      uint64_t  sas_maxrto;      /* Maximum Observed RTO this period */
      uint64_t  sas_outseqtsns;  /* TSN received > next expected */
      uint64_t  sas_osacks;      /* SACKs sent */
      uint64_t  sas_isacks;      /* SACKs received */
      uint64_t  sas_octrlchunks; /* Control chunks sent - no dups */
      uint64_t  sas_ictrlchunks; /* Control chunks received - no dups */
      uint64_t  sas_oodchunks;   /* Ordered data chunks sent */
      uint64_t  sas_iodchunks;   /* Ordered data chunks received */
      uint64_t  sas_ouodchunks;  /* Unordered data chunks sent */
      uint64_t  sas_iuodchunks;  /* Unordered data chunks received */
      uint64_t  sas_idupchunks;  /* Dups received (ordered+unordered) */
} sctp_assoc_stats_t;
```

Multihoming The ability of SCTP to use multiple addresses in an association can create issues with some network utilities. This requires a system administrator to be careful in setting up the system.

For example, the `tcpd` allows an administrator to use a simple form of address/hostname access control. While `tcpd` can work with SCTP, the access control part can have some problems. The `tcpd` access control is only based on one of the addresses at association setup time. Once as association is allowed, no more checking is performed. This means that during the life time of the association, SCTP packets from different addresses of the peer host can be received in the system. This may not be what the system administrator wants as some of the peer's addresses are supposed to be blocked.

Another example is the use of IP Filter, which provides several functions such as IP packet filtering (`ipf(1M)`) and NAT `ipnat(1M)`). For packet filtering, one issue is that a filter policy can block packets from some of the addresses of an association while allowing packets from other addresses to go through. This can degrade SCTP's performance when failure occurs. There is a more serious issue with IP address rewrite by NAT. At association setup time, SCTP endpoints exchange IP addresses. But IP Filter is not aware of this. So when NAT is done on a packet, it may change the address to an unacceptable one. Thus the SCTP association setup may succeed but packets cannot go through afterwards when a different IP address is used for the association.

See Also `ipf(1M)`, `ipnat(1M)`, `ndd(1M)`, `ioctl(2)`, `close(2)`, `read(2)`, `write(2)`, `accept(3SOCKET)`, `bind(3SOCKET)`, `connect(3SOCKET)`, `getprotobyname(3SOCKET)`, `getsockopt(3SOCKET)`, `libsctp(3LIB)`, `listen(3SOCKET)`, `recv(3SOCKET)`, `recvfrom(3SOCKET)`, `recvmsg(3SOCKET)`, `sctp_bindx(3SOCKET)`, `sctp_getladdrs(3SOCKET)`, `sctp_getpaddrs(3SOCKET)`, `sctp_freepaddrs(3SOCKET)`, `sctp_opt_info(3SOCKET)`, `sctp_peeloff(3SOCKET)`, `sctp_recvmsg(3SOCKET)`, `sctp_sendmsg(3SOCKET)`, `send(3SOCKET)`, `sendmsg(3SOCKET)`, `sendto(3SOCKET)`, `socket(3SOCKET)`, `ipfilter(5)`, `tcp(7P)`, `udp(7P)`, `inet(7P)`, `inet6(7P)`, `ip(7P)`, `ip6(7P)`

R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zang, V. Paxson, *RFC 2960, Stream Control Transmission Protocol*, October 2000

L. Ong, J. Yoakum, *RFC 3286, An Introduction to Stream Control Transmission Protocol (SCTP)*, May 2002

J. Stone, R. Stewart, D. Otis, *RFC 3309, Stream Control Transmission Protocol (SCTP) Checksum Change*, September 2002.

Diagnostics A socket operation may fail if:

`EPROTONOSUPPORT`	The socket type is other than `SOCK_STREAM` and `SOCK_SEQPACKET`.
`ETIMEDOUT`	An association was dropped due to excessive retransmissions.
`ECONNREFUSED`	The remote peer refused establishing an association.
`EADDRINUSE`	A `bind()` operation was attempted on a socket with a network address/port pair that has already been bound to another socket.

EINVAL	A `bind()` operation was attempted on a socket with an invalid network address.
EPERM	A `bind()` operation was attempted on a socket with a “reserved“ port number and the effective user ID of the process was not the privileged user.

Name sd – SCSI disk and ATAPI/SCSI CD-ROM device driver

Synopsis `sd@target,lun:partition`

Description To open a device without checking if the vtoc is valid, use the O_NDELAY flag. When the device is opened using O_NDELAY, the first read or write to the device that happens after the open results in the label being read if the label is not currently valid. Once read, the label remains valid until the last close of the device. Except for reading the label, O_NDELAY has no impact on the driver.

SPARC The `sd` SCSI and SCSI/ATAPI driver supports embedded SCSI-2 and CCS-compatible SCSI disk and CD-ROM drives, ATAPI 2.6 (SFF-8020i)-compliant CD-ROM drives, SFF-8090–compliant SCSI/ATAPI DVD-ROM drives, IOMEGA SCSI/ATAPI ZIP drives, SCSI JAZ drives, and USB mass storage devices (refer to `scsa2usb(7D)`).

To determine the disk drive type, use the SCSI/ATAPI inquiry command and read the volume label stored on block 0 of the drive. (The volume label describes the disk geometry and partitioning and must be present for the disk to be mounted by the system.) A volume label is not required for removable, re-writable or read-only media.

x86 Only The `sd`driver supports embedded SCSI-2 and CCS-compatible SCSI disk and CD-ROM drives, ATAPI 2.6 (SFF-8020i)-compliant CD-ROM drives, SFF-8090-compliant SCSI/ATAPI DVD-ROM drives, IOMEGA SCSI/ATAPI ZIP drives, and SCSI JAZ drives.

The x86 BIOS legacy requires a master boot record (MBR) and `fdisk` table in the first physical sector of the bootable media. If the x86 hard disk contains a Solaris disk label, it is located in the second 512-byte sector of the FDISK partition.

Device Special Files Block-files access the disk using normal buffering mechanism and are read-from and written-to without regard to physical disk records. A `raw` interface enables direct transmission between the disk and the user's read or write buffer. A single `read` or `write` call usually results in a single I/O operation, therefore raw I/O is more efficient when many bytes are transmitted. Block files names are found in `/dev/dsk`; raw file names are found in `/dev/rdsk`.

I/O requests to the raw device must be aligned on a 512-byte (`DEV_BSIZE`) boundary and all I/O request lengths must be in multiples of 512 bytes. Requests that do not meet these requirements will trigger an `EINVAL` error. There are no alignment or length restrictions on I/O requests to the block device.

Cd-ROM Drive Support A CD-ROM disk is single-sided and contains approximately 640 megabytes of data or 74 minutes of audio. When the CD-ROM is opened, the eject button is disabled to prevent manual removal of the disk until the last `close()` is called. No volume label is required for a CD-ROM. The disk geometry and partitioning information are constant and never change. If the CD-ROM contains data recorded in a Solaris-aware file system format, it can be mounted using the appropriate Solaris file system support.

Dvd-ROM Drive Support

DVD-ROM media can be single or double-sided and can be recorded upon using a single or double layer structure. Double-layer media provides parallel or opposite track paths. A DVD-ROM can hold from between 4.5 Gbytes and 17 Gbytes of data, depending on the layer structure used for recording and if the DVD-ROM is single or double-sided.

When the DVD-ROM is opened, the eject button is disabled to prevent the manual removal of a disk until the last `close()` is called. No volume label is required for a DVD-ROM. If the DVD-ROM contains data recorded in a Solaris-aware file system format, it can be mounted using the appropriate Solaris file system support.

Zip/JAZ Drive Support

ZIP/JAZ media provide varied data capacity points; a single JAZ drive can store up to 2 GBytes of data, while a ZIP-250 can store up to 250MBytes of data. ZIP/JAZ drives can be read-from or written-to using the appropriate drive.

When a ZIP/JAZ drive is opened, the eject button is disabled to prevent the manual removal of a disk until the last `close()` is called. No volume label is required for a ZIP/JAZ drive. If the ZIP/JAZ drive contains data recorded in a Solaris-aware file system format, it can be mounted using the appropriate Solaris file system support.

Device Statistics Support

Each device maintains I/O statistics for the device and for partitions allocated for that device. For each device/partition, the driver accumulates reads, writes, bytes read, and bytes written. The driver also initiates hi-resolution time stamps at queue entry and exit points to enable monitoring of residence time and cumulative residence-length product for each queue.

Not all device drivers make per-partition IO statistics available for reporting. `sd` and `ssd(7D)` per-partition statistics are enabled by default but may disabled in their configuration files.

ioctls

Refer to `dkio(7I)`, and `cdio(7I)`

ERRORS

`EACCES`	Permission denied
`EBUSY`	The partition was opened exclusively by another thread
`EFAULT`	The argument features a bad address
`EINVAL`	Invalid argument
`ENOTTY`	The device does not support the requested ioctl function
`ENXIO`	During opening, the device did not exist. During close, the drive unlock failed
`EROFS`	The device is read-only
`EAGAIN`	Resource temporarily unavailable
`EINTR`	A signal was caught during the execution of the `ioctl()` function
`ENOMEM`	Insufficient memory
`EPERM`	Insufficent access permission

EIO	An I/O error occurred. Refer to notes for details on copy-protected DVD-ROM media.

Configuration The sd driver can be configured by defining properties in the sd.conf file. The sd driver supports the following properties:

enable-partition-kstats	The default value is 1, which causes partition IO statistics to be maintained. Set this value to zero to prevent the driver from recording partition statistics. This slightly reduces the CPU overhead for IO, mimimizes the amount of sar(1) data collected and makes these statistics unavailable for reporting by iostat(1M) even though the -p/-P option is specified. Regardless of this setting, disk IO statistics are always maintained.
qfull-retries	The supplied value is passed as the qfull-retries capability value of the HBA driver. See scsi_ifsetcap(9F) for details.
qfull-retry-interval	The supplied value is passed as the qfull-retry interval capability value of the HBA driver. See scsi_ifsetcap(9F) for details.
allow-bus-device-reset	The default value is 1, which allows resetting to occur. Set this value to 0 (zero) to prevent the sd driver from calling scsi_reset(9F) with a second argument of RESET_TARGET when in error-recovery mode. This scsi_reset(9F) call may prompt the HBA driver to send a SCSI Bus Device Reset message. The scsi_reset(9F) call with a second argument of RESET_TARGET may result from an explicit request via the USCSICMD ioctl. Some high-availability multi-initiator systems may wish to prohibit the Bus Device Reset message; to do this, set the allow-bus-device-reset property to 0.
optical-device-bind	Controls the binding of the driver to non self-identifying SCSI target optical devices. (See scsi(4)). The default value is 1, which causes sd to bind to DTYPE_OPTICAL devices (as noted in scsi(4)). Setting this value to 0 prevents automatic binding. The default behavior for the SPARC-based sd driver prior to Solaris 9 was not to bind to optical devices.
power-condition	Boolean type, when set to False, it indicates that the disk does not support power condition field in the START STOP UNIT command.

In addition to the above properties, some device-specific tunables can be configured in sd.conf using the sd-config-list global property. The value of this property is a list of duplets. The formal syntax is:

```
sd-config-list = <duplet> [, <duplet> ]* ;

where

<duplet>:= "<vid+pid>" , "<tunable-list>"

and

<tunable-list>:= <tunable>  [, <tunable> ]*;
<tunable> = <name> : <value>

The <vid+pid> is the string that is returned by the target device
on a SCSI inquiry command.

The <tunable-list> contains one or more tunables to apply to
all target devices with the specified <vid+pid>.

Each <tunable> is a <name> : <value> pair. Supported
tunable names are:

   delay-busy: when busy, nsecs of delay before retry.

   retries-timeout: retries to perform on an IO timeout.
```

`mmc-gesn-polling` For optical drives compliant with `MMC-3` and supporting the `GET EVENT STATUS NOTIFICATION` command, this command is used for periodic media state polling, usually initiated by the `DKIOCSTATE` `dkio(7I)` ioctl. To disable the use of this command, set this boolean property to `false`. In that case, either the `TEST UNIT READY` or zero-length `WRITE(10)` command is used instead.

Examples

`The following is an example of a global sd-config-list property:`

```
sd-config-list =
   "SUN     T4", "delay-busy:600, retries-timeout:6",
   "SUN     StorEdge_3510", "retries-timeout:3";
```

Files

`/kernel/drv/sd.conf`	Driver configuration file
`/dev/dsk/cntndnsn`	Block files
`/dev/rdsk/cntndnsn`	Raw files

Where:

cn controller n

tn SCSI target id n (0-6)

dn SCSI LUN n (0-7 normally; some HBAs support LUNs to 15 or 32. See the specific manpage for details)

sn partition n (0-7)

x86 Only `/dev/rdsk/cntndnpn` raw files

Where:

pn Where *n*=0 the node corresponds to the entire disk.

See Also `sar(1)`, `cfgadm_scsi(1M)`, `fdisk(1M)`, `format(1M)`, `iostat(1M)`, `close(2)`, `ioctl(2)`, `lseek(2)`, `read(2)`, `write(2)`, `driver.conf(4)`, `scsi(4)`, `filesystem(5)`, `scsa2usb(7D)`, `ssd(7D)`, `hsfs(7FS)`, `pcfs(7FS)`, `udfs(7FS)`, `cdio(7I)`, `dkio(7I)`, `scsi_ifsetcap(9F)`, `scsi_reset(9F)`

ANSI Small Computer System Interface-2 (SCSI-2)

ATA Packet Interface for CD-ROMs, SFF-8020i

Mt.Fuji Commands for CD and DVD, SFF8090v3

http://*www.sun.com/io*

Diagnostics

`Error for Command:`<*command name*>
`Error Level: Fatal`
`Requested Block:` <*n*>
`Error Block:` <*m*>
`Vendor:'`<*vendorname*>`'`
`Serial Number:'`<*serial number*>`'`
`Sense Key:`<*sense key name*>

ASC: 0x<a> (<ASC name>), ASCQ: 0x<b>, FRU: 0x<c>
The command indicated by <command name> failed. The Requested Block is the block where the transfer started and the Error Block is the block that caused the error. Sense Key, ASC, and ASCQ information is returned by the target in response to a request sense command.

Caddy not inserted in drive
The drive is not ready because no caddy has been inserted.

Check Condition on REQUEST SENSE
A REQUEST SENSE command completed with a check condition. The original command will be retried a number of times.

Label says <m> blocks Drive says <n> blocks
There is a discrepancy between the label and what the drive returned on the `READ CAPACITY` command.

Not enough sense information
The request sense data was less than expected.

Request Sense couldn't get sense data
The `REQUEST SENSE` command did not transfer any data.

Reservation Conflict
The drive was reserved by another initiator.

SCSI transport failed: reason `'xxxx'`: {retrying|giving up}
The host adapter has failed to transport a command to the target for the reason stated. The driver will either retry the command or, ultimately, give up.

Unhandled Sense Key<*n*>
The REQUEST SENSE data included an invalid sense.

Unit not ready. Additional sense code 0x
<*n*> The drive is not ready.

Can't do switch back to mode 1
A failure to switch back to read mode 1.

Corrupt label - bad geometry
The disk label is corrupted.

Corrupt label - label checksum failed
The disk label is corrupted.

Corrupt label - wrong magic number
The disk label is corrupted.

Device busy too long
The drive returned busy during a number of retries.

Disk not responding to selection
The drive is powered down or died

Failed to handle UA
A retry on a Unit Attention condition failed.

I/O to invalid geometry
The geometry of the drive could not be established.

Incomplete read/write - retrying/giving up
There was a residue after the command completed normally.

No bp for direct access device format geometry
A bp with consistent memory could not be allocated.

No bp for disk label
A bp with consistent memory could not be allocated.

No bp for fdisk
A bp with consistent memory could not be allocated.

No bp for rigid disk geometry
A bp with consistent memory could not be allocated.

No mem for property
Free memory pool exhausted.

No memory for direct access device format geometry
Free memory pool exhausted.

No memory for disk label
Free memory pool exhausted.

No memory for rigid disk geometry
The disk label is corrupted.

No resources for dumping
A packet could not be allocated during dumping.

Offline
Drive went offline; probably powered down.

Requeue of command fails
Driver attempted to retry a command and experienced a transport error.

sdrestart transport failed()
Driver attempted to retry a command and experienced a transport error.

Transfer length not modulo
Illegal request size.

Transport of request sense fails()
Driver attempted to submit a request sense command and failed.

Transport rejected()
Host adapter driver was unable to accept a command.

Unable to read label
Failure to read disk label.

Unit does not respond to selection
Drive went offline; probably powered down.

Notes DVD-ROM media containing DVD-Video data may follow/adhere to the requirements of content scrambling system or copy protection scheme. Reading of copy-protected sector will cause I/O error. Users are advised to use the appropriate playback software to view video contents on DVD-ROM media containing DVD-Video data.

Name sda – SD/MMC architecture

Description The `sda` module provides support services for Secure Digital (SD) and MultiMediaCard (MMC) slot and card device drivers.

Files

`/kernel/misc/sda`	32-bit ELF kernel module (x86)
`/kernel/misc/amd64/sda`	64-bit ELF kernel module (x86)

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWsdcard

See Also `cfgadm_sdcard(1M)`, `attributes(5)`, `blk2scsa(7D)`, `sd(7D)`, `sdcard(7D)`, `sdhost(7D)`, `scsa2usb(7D)`

System Administration Guide, Volume I

System Administration Guide: Basic Administration

Notes The `sda` module provides support only for SD/MMC devices that are connected via a supported slot driver. Notably, slots that are on USB busses are normally treated as USB mass storage devices and are serviced by the `scsa2usb(7D)` driver.

Name SDC – System Duty Cycle scheduling class

Description The System Duty Cycle (`SDC`) scheduling class is used for some CPU-intensive kernel thread workloads. Like the `SYS` class, it cannot be used for user processes.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Private

See Also `attributes(5)`

Name sdcard – SD/MMC memory card driver

Description The `sdcard` memory card driver supports MultiMediaCard (MMC), Secure Digital (SD), and Secure Digital High Capacity (SDHC) memory cards. It uses the `blk2scsa(7D)` service to present these cards to the system as SCSI disks on a virtual SCSI bus, creating a child device to be serviced with `sd(7D)`. Each card appears as its own SCSI LUN. Cards are hot-pluggable and removable.

Device Special Files Disk block special file names are located in `/dev/dsk`. Raw file names are located in `/dev/rdsk`. See `sd(7D)`.

ioctls See `dkio(7I)`

Errors See `sd(7D)` and `blk2scsa(7D)`. Additionally, `sdcard` may issue the following warnings, which indicate a failure to identify the card as a supported type:

```
"Unknown SD CSD version (%d)"
"Unknown MMC CSD version (%d)"
"Unknown MMCA version (%d)"
"Card type unknown"
```

Files Device special files for the storage device are created in the same way as those for a SCSI disk. See `sd(7D)` for more information.

/dev/dsk/*cntndnsn*	Block files for disks
/dev/rdsk/*cntndnsn*	Raw files for disks
/kernel/drv/sdcard	32-bit ELF kernel module (x86)
/kernel/misc/amd64/sdcard	64-bit ELF kernel module (x86)

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWsdcard

See Also `rmformat(1)`, `rmmount(1)`, `cfgadm_scsi(1M)`, `cfgadm_sdcard(1M)`, `fdisk(1M)`, `mount(1M)`, `umount(1M)`, `scsi(4)`, `vfstab(4)`, `attributes(5)`, `blk2scsa(7D)`, `sd(7D)`, `sda(7D)`, `dkio(7I)`, `pcfs(7FS)`

System Administration Guide, Volume I

System Administration Guide: Basic Administration

Diagnostics See sd(7D).

Name sdhost – Standard-compliant Secure Digital slot driver

Synopsis

```
pciclass,080500@unit-address
pciclass,080501@unit-address
```

Description The `sdhost` driver supports Secure Digital (SD) standard media slots commonly found on mobile computers.

Files Memory card device files are created by the `sdcard(7D)` driver. An attachment point device file is created for each physical slot on the system:

`/dev/sdcardx/y`	Attachment point for slot *y* on controller *x*. Typically this is named `/dev/sdcard0/0`.
`/kernel/drv/sdhost`	32-bit ELF kernel module (x86).
`/kernel/drv/amd64/sdhost`	64-bit ELF kernel module (x86).

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWsdcard

See Also `cfgadm_sdcard(1M)`, `attributes(5)`, `sda(7D)`, `sdcard(7D)`

System Administration Guide, Volume I

System Administration Guide: Basic Administration

Name sdp – Sockets Direct Protocol driver

Synopsis

```
#include <socket.h>

#include <netinet/in.h>

s = socket(AF_INET, SOCK_STREAM, PROTO_SDP);

s = socket(AF_INET6, SOCK_STREAM, PROTO_SDP);
```

Description The Sockets Direct Protocol (SDP) is a transport protocol layered over the *Infiniband Transport Framework* (IBTF). SDP is a standard implementation based on Annex 4 of the *Infiniband Architecture Specification Vol 1* and provides reliable byte-stream, flow controlled two-way data transmission that closely mimics the Transmission Control Protocol (TCP).

SDP supports a sockets-based SOCK_STREAM interface to application programs. It also supports graceful close (including half-closed sockets), IP addressing (IPv4 or IPv6), the connecting/accepting connect model, out-of-band (OOB) data and common socket options. The SDP protocol also supports kernel bypass data transfers and data transfers from send-upper-layer-protocol (ULP) buffers to receive ULP buffers. A SDP message includes a BSDH header followed by data. (A BSDH header advertises the amount of available buffers on the local side).

SDP networking functionality is broken into the `sdp` driver and a function call-based sockfs implementation. A new protocol family of PROTO_SDP is introduced to use the SDP transport provided by the driver.

Sockets utilizing SDP are either active or passive. Active sockets initiate connections to passive sockets. Both active and passive sockets must have their local IP or IPv6 address and SDP port number bound with the `bind(3SOCKET)` system call after the socket is created. By default, SDP sockets are active. A passive socket is created by calling the `listen(3SOCKET)` system call after binding the socket with bind(). This process establishes a queueing parameter for the passive socket. Connections to the passive socket can be received with the `accept(3SOCKET)` system call. Active sockets use the `connect(3SOCKET)` call after binding to initiate connections.

In most cases, SDP sends data when it is presented. When outstanding data is not yet acknowledged, SDP gathers small amounts of output to be sent in a single packet once an acknowledgement is received. For a small number of clients this packetization may cause significant delays. To circumvent this problem, SDP provided by the driver supplies SDP_NODELAY, a socket-level boolean option. Note that this behavior is similar to the TCP_NODELAY option.

SDP provides an urgent data mechanism that can be invoked using the out-of-band provisions of `send(3SOCKET)`. The out-of-band delivery behavior is identical to TCP. The caller may mark one byte as "urgent" with the MSG_OOB flag to `send(3SOCKET)`. This sets an "urgent pointer" pointing to the byte in the SDP stream. The receiver of the stream is notified of the urgent data by a `SIGURG` signal. The SIOCATMARK `ioctl(2)` request returns a

value indicating whether the stream is at the urgent mark. Because the system never returns data across the urgent mark in a single `read(2)` call, it is possible to advance to the urgent data in a simple loop which reads data, testing the socket with the `SIOCATMARK` ioctl() request until it reaches the mark.

Address Formats SDP uses IP/IPv6 addresses to refer to local and remote devices and opens a reliable connected IB connection between two end points. The `sdp` driver supports a point-to-point connection, however broadcasting and multicasting are not supported.

Socket Options SDP supports setsockopt and getsockopt to set and read socket options. Very few socket options affect SDP protocol operations. Other common socket options are processed but do not affect SDP protocol operation. All socket options are checked for validity. A getsockopt returns the values set or toggled by setsockopt. Socket options that affect protocol operations are SO_LINGER, SO_DEBUG, SO_REUSEADDR and SO_OOBINLINE.

Errors

`EISCONN`	A connect() operation was attempted on a socket on which a connect() operation had already been performed.
`ECONNRESET`	The remote peer forced the connection to be closed. This usually occurs when the remote machine loses state information about the connection due to a crash.
`ECONNREFUSED`	The remote peer actively refused connection establishment. This usually occurs because no process is listening to the port.
`EADDRINUSE`	A bind() operation was attempted on a socket with a network address/port pair that has already been bound to another socket.
`EADDRNOTAVAIL`	A bind() operation was attempted on a socket with a network address for which no network interface exists.
`EACCES`	A bind() operation was attempted with a reserved port number and the effective user ID of the process was not the privileged user.
`ENOBUFS`	The system ran out of memory for internal data structures.

Files

`/kernel/drv/sdp`
32–bit ELF kernel module (x86).

`/kernel/drv/amd64/sdp`
64–bit ELF kernel module (x86).

`/kernel/drv/sparcv9/sdp`
64–bit ELF kernel module (SPARC).

`/kernel/drv/sdpib`
32–bit ELF kernel module (x86).

`/kernel/drv/amd64/sdpib`
64–bit ELF kernel module (x86).

`/kernel/drv/sparcv9/sdpib`
64–bit ELF kernel module (SPARC).

Attributes See `attributes(5)` for descriptions of the following attribute:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86, SPARC
Availability	SUNWibsdp, SUNWibsdp

See Also `read(2)`, `getsockopt(3XNET)`, `socket.h(3HEAD)`, `accept(3SOCKET)`, `bind(3SOCKET)`, `connect(3SOCKET)`, `send(3SOCKET)`, `attributes(5)`, `standards(5)`

Infiniband Architecture Specification Vol 1– Annex 4 — November, 2002

Name sdt – DTrace statically defined tracing provider

Description The `sdt` driver is a DTrace dynamic tracing provider that performs dynamic instrumentation at statically-defined locations in the Solaris kernel.

The `sdt` provider allows kernel developers to explicitly create probes at formally designated locations in the operating system kernel and loadable modules, allowing the implementor to consciously choose the points in their code that are desired probe points, and to convey some semantic knowledge about that point with the choice of probe name and a relevant set of arguments.

The `sdt` driver is not a public interface and you access instrumentation offered by this provider through DTrace. Refer to the *Solaris Dynamic Tracing Guide* for a description of the public documented interfaces available for the DTrace facility and the probes offered by the `sdt` provider.

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWdtrp
Interface Stability	Private

See Also `dtrace(1M)`, `attributes(5)`, `dtrace(7D)`

Solaris Dynamic Tracing Guide

Name se – Siemens 82532 ESCC serial communications driver

Synopsis `se@`*bus_address*`:`*port_name*`[,cu]`

Description The `se` module is a loadable STREAMS driver that provides basic support for the 82532 ESCC hardware and basic asynchronous and synchronous communication support. This manual page describes the asynchronous protocol interface; for information on the synchronous interface, please see the `se_hdlc(7D)` manual page.

This module is affected by the setting of specific eeprom variables. For information on parameters that are persistent across reboots, see the `eeprom(1M)` man page.

The platform specific device bus address for the `se` module is *bus_address*. The `se` module's *port_name* is a single letter (a-z).

Note – During boot up, ttya/b characteristics are read from the `/kernel/drv/options.conf` file and changed from the PROM defaults to reflect Solaris defaults. Messages displayed on the console after this point are based on settings in that file. If you switch a characteristic, (for example, the baud rate of the console terminal), you must revise the `/kernel/drv/options.conf` or the console will be configured to an unusable configuration and console messages will be garbled by the mismatched serial port settings.

Application Programming Interface The Siemens 82532 provides two serial input/output channels capable of supporting a variety of communication protocols. A typical system will use one of these devices to implement two serial ports (*port_name*), usually configured for RS-423 (which also supports most RS-232 equipment). The Siemens 82532 uses 64 character input and output FIFOs to reduce system overhead. When receiving characters, the CPU is notified when 32 characters have arrived (one-half of receive buffer is full) or no character has arrived in the time it would take to receive four characters at the current baud rate.

When sending characters, the Siemens 82532 places the first 64 characters to be sent into its output FIFO and then notifies the CPU when it is half empty (32 characters left). Because the `se` module waits for the Siemens 82532 to transmit the remaining characters within its output FIFO before making requested changes, delays may occur when the port's attributes are being modified.

The `se` module implements CTS/RTS flow control in hardware. To prevent data overruns, remove CTS/RTS flow control responsibility from the CPU during periods of high system load.

In `async` mode (obtained by opening `/dev/cua/[`*a-z*`]`, `/dev/term/[`*a-z*`]` or `/dev/tty[`*a-z*`]`), the driver supports the `termio(7I)` device control functions specified by flags in the `c_cflag` word of the `termios` structure, and by the `IGNBRK, IGNPAR, PARMRK,` or `INPCK` flags in the `c_iflag` word. All other `termio(7I)` functions must be performed by STREAMS modules pushed atop the driver. When a device is opened, the `ldterm(7M)` and `ttcompat(7M)` STREAMS modules are automatically pushed on top of the stream, providing the standard `termio` interface.

You can connect a single tty line to a modem for incoming and outgoing calls using a special feature controlled by the minor device number. By accessing character-special devices with names of the form `/dev/cua/`[*a-z*], it is possible to open a port without the `Carrier Detect` signal being asserted, either through hardware or an equivalent software mechanism. These devices are commonly known as dial-out lines.

After a `/dev/cua/`[*a-z*] line is opened, the corresponding tty line cannot be opened until the `/dev/cua/`*[a-z]* line is closed. A blocking open will wait until the `/dev/cua/`[*a-z*] line is closed (which will drop `Data Terminal Ready` and `Carrier Detect`) and carrier is detected again. A non-blocking open will return an error. If the tty line has been opened successfully (usually only when carrier is recognized on the modem), the corresponding `/dev/cua/`[*a-z*] line cannot be opened. This allows a modem to be attached to a device, (for example, `/dev/term/`[*a-z*] renamed from `/dev/tty`[*a-z*]) and used for dial-in (by enabling the line for login in `/etc/inittab`) and dial-out (by `tip(1)` or `uucp(1C)`) as `/dev/cua/`[*a-z*] when no one is logged in on the line.

The `se` driver can be configured to support mice as well as applications requiring no buffering on the receive fifo. You can do this in one of two ways:

1. Using the `se` configuration file — To configure the `se` device to support mice using this approach, create an `se.conf` under `/kernel/drv`, then add keywords of the form: `disable-rfifo-port<a/b><`*instance number*>. For example, if your system has two `se` devices and you want port *b* on the device (associated with instance *0*) and port *a* (associated with instance *1*) to have their receive fifo disabled, the `se.conf` file must contain the following entries:

   ```
   disable-rfifo-portb0;
   disable-rfifo-porta1;
   ```

2. Programmatically — You can also configure the `se` device to support mice programatically by using the SERVICEIMM stream call to turn buffering off on the receive fifo, and/or SERVICEDEF to turn it back on again.

ioctls The `se` module supports the standard set of `termio ioctl( )` calls.

Breaks can be generated by the `TCSBRK`, `TIOCSBRK`, and `TIOCCBRK ioctl()` calls.

The state of the `DCD, CTS, RTS,` and `DTR` interface signals can be queried through the use of the `TIOCM_CAR, TIOCM_CTS, TIOCM_RTS,` and `TIOCM_DTR` arguments to the `TIOCMGET ioctl` command, respectively. Due to hardware limitations, only the `RTS` and `DTR` signals may be set through their respective arguments to the `TIOCMSET, TIOCMBIS,` and `TIOCMBIC ioctl` commands.

The input and output line speeds may be set to all baud rates supported by `termio`. Input and output line speeds cannot be set independently; when you set the output speed, the input speed is automatically set to the same speed.

When using baud rates over 100,000 baud, the software changes the line driver configuration to handle the higher data rates. This action decreases the theoretical maximum cable length from 70 meters to 30 meters.

When the `se` module is used to service the serial console port, it supports a BREAK condition that allows the system to enter the debugger or the monitor. The BREAK condition is generated by hardware and it is usually enabled by default. A BREAK condition originating from erroneous electrical signals cannot be distinguished from one deliberately sent by remote DCE. Due to a risk of incorrect sequence interpretation, SLIP and certain other binary protocols should not be run over the serial console port when Alternate Break sequence is in effect. Although PPP is a binary protocol, it is able to avoid these sequences using the ACCM feature in *RFC 1662*. For Solaris PPP 4.0, you do this by adding the following line to the `/etc/ppp/options` file (or other configuration files used for the connection; see `pppd(1M)` for details):

```
asyncmap  0x00002000
```

By default, the Alternate Break sequence is a three character sequence: carriage return, tilde and control-B (CR ~ CTRL-B), but may be changed by the driver. For information on breaking (entering the debugger or monitor), see `kmdb(1)` and `kb(7M)`.

Errors An `open()` will fail under the following conditions:

`ENXIO` The unit being opened does not exist.

`EBUSY` The dial-out device is being opened and the dial-in device is already open, or the dial-in device is being opened with a no-delay open and the dial-out device is already open.

`EBUSY` The port is in use by another serial protocol.

`EBUSY` The unit has been marked as exclusive-use by another process with a `TIOCEXCL ioctl( )` call.

`EINTR` The open was interrupted by the delivery of a signal.

Files

`/dev/cua/`*`[a-z]`*	dial-out tty lines
`/dev/term/[a-z]`	dial-in tty lines
`/dev/tty`*`[a-z]`*	binary compatibility package device names
`/dev/se_hdlc`*`[0-9]`*	synchronous devices - see `se_hdlc(7D)`.
`/dev/se_hdlc`	synchronous control clone device
`/kernel/drv/options.conf`	System wide default device driver properties

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC

See Also kmdb(1), tip(1), cu(1C), uucp(1C), eeprom(1M), ports(1M), pppd(1M), ioctl(2), open(2), attributes(5),zs(7D), zsh(7D), se_hdlc(7D), termio(7I), ldterm(7M), ttcompat(7M), kb(7M)

Diagnostics

`se`*n* `: fifo overrun` The Siemens 82532 internal FIFO received more data than it could handle. This indicates that Solaris was not servicing data interrupts fast enough and suggests a system with too many interrupts or a data line with a data rate that is too high.

`se`*n* `: buffer overrun` The `se` module was unable to store data it removed from the Siemens 82532 FIFO. The user process is not reading data fast enough, and suggests an overloaded system. If possible, the application should enable flow control (either CTSRTS or XONXOFF) to allow the driver to backpressure the remote system when the local buffers fill up.

Name se_hdlc – on-board high-performance serial HDLC interface

Synopsis `se@`*bus_address*`:`*port_number*`[, hdlc]`

Description The `se_hdlc` devices are a synchronous hdlc-framing interface for the se serial devices. Both built-in serial ports (*port_number*) on platforms which have the `se` serial devices, support synchronous data transfer at a maximum rate of 384 kbps. *bus_address* is the platform specific `se` device bus address. *port_number* is a single digit number (0-9).

Application Programming Interface The `se_hdlc`*n* devices provide a data path which supports the transfer of data via `read(2)` and `write(2)` system calls, as well as `ioctl(2)` calls. Data path opens are exclusive in order to protect against injection or diversion of data by another process.

The `se_hdlc` device provides a separate control path for use by programs that need to configure or monitor a connection independent of any exclusive access restrictions imposed by data path opens. Up to three control paths may be active on a particular serial channel at any one time. Control path accesses are restricted to `ioctl(2)` calls only; no data transfer is possible.

When used in synchronous modes, the SAB 82532 ESCC supports several options for clock sourcing and data encolding. Both the transmit and receive clock sources can be set to be the external Transmit clock (TRxC), external Receive Clock (RTxC), the internal Baud Rate Generator (BRG), or the output of the ESCC 's Digital Phase-Lock Loop (DPLL).

The BRG is a programmable divisor that derives a clock frequency from the PCLK input signal to the ESCC. The programmed baud rate is translated into a floating point (6-bit mantissa, 4–bit exponent) number time constant that is stored in the ESCC.

A local loopback mode is available, primarily for use by `syncloop(1M)` for testing purposes, and should not be confused with SDLC loop mode, which is not supported on this interface. Also, an auto-echo feature may be selected that causes all incoming data to be routed to the transmit data line, allowing the port to act as the remote end of a digital loop. Neither of these options should be selected casually, or left in use when not needed.

The `se` driver keeps running totals of various hardware generated events for each channel. These include numbers of packets and characters sent and received, abort conditions detected by the receiver, receive CRC errors, transmit underruns, receive overruns, input errors and output errors, and message block allocation failures. Input errors are logged whenever an incoming message must be discarded, such as when an abort or CRC error is detected, a receive overrun occurs, or when no message block is available to store incoming data. Output errors are logged when the data must be discarded due to underruns, CTS drops during transmission, CTS timeouts, or excessive watchdog timeouts caused by a cable break.

ioctls The `se` driver supports the following `ioctl()` commands.

`S_IOCGETMODE` Return a `struct scc_mode` containing parameters currently in use. These include the transmit and receive clock sources, boolean loopback and NRZI mode flags and the integer baud rate.

`S_IOCSETMODE` The argument is a `struct scc_mode` from which the ESCC channel will be programmed.

`S_IOCGETSTATS` Return a `struct sl_stats` containing the current totals of hardware-generated events. These include numbers of packets and characters sent and received by the driver, aborts and CRC errors detected, transmit underruns, and receive overruns.

`S_IOCCLRSTATS` Clear the hardware statistics for this channel.

`S_IOCGETSPEED` Returns the currently set baud rate as an integer. This may not reflect the actual data transfer rate if external clocks are used.

`S_IOCGETMCTL` Returns the current state of the CTS and DCD incoming modem interface signals as an integer.

The following structures are used with `se hdlc ioctl()` commands:

```
struct scc_mode {
 char sm_txclock;    /* transmit clock sources */
 char sm_rxclock;    /* receive clock sources */
 char sm_iflags;     /* data and clock inversion flags (non-zsh) */
 uchar_t sm_config;  /* boolean configuration options */
 int sm_baudrate;    /* real baud rate */
 int sm_retval;      /* reason codes for ioctl failures */
};
struct sl_stats {
 long    ipack;        /* input packets */
 long    opack;        /* output packets */
 long    ichar;        /* input bytes */
 long    ochar;        /* output bytes */
 long    abort;        /* abort received */
 long    crc;          /* CRC error */
 long    cts;          /* CTS timeouts */
 long    dcd;          /* Carrier drops */
 long    overrun;      /* receive overrun */
 long    underrun;     /* transmit underrun */
 long    ierror;       /* input error */
 long    oerror;       /* output error */
 long    nobuffers;    /* receive side memory allocation failure */
};
```

Errors An `open()` will fail if a STREAMS message block cannot be allocated or under the following conditions:

`ENXIO` The unit being opened does not exist.

`EBUSY` The device is in use by another serial protocol.

An `ioctl()` will fail under the following conditions:

`EINVAL` An attempt was made to select an invalid clocking source.

`EINVAL` The baud rate specified for use with the baud rate generator would translate to a null time constant in the ESCC's registers.

Files

`/dev/se_hdlc[0-1], /dev/se_hdlc`	character-special devices
`/usr/include/sys/ser_sync.h`	header file specifying synchronous serial communication definitions

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC

See Also `syncinit(1M)`, `syncloop(1M)`, `syncstat(1M)`, `ioctl(2)`, `open(2)`, `read(2)`, `write(2)`, `attributes(5)`, `se(7D)`, `zsh(7D)`

Siemens ESCC2 SAB 82532 Enhanced Serial Communication Controller User's Manual

Diagnostics

se_hdlc clone open failed, no memory, rq=*nnn*
A kernel memory allocation failed for one of the private data structures. The value of *nnn* is the address of the read queue passed to `open(2)`.

se_hdlc: clone device must be attached before use!
An operation was attempted through a control path before that path had been attached to a particular serial channel.

se_hdlc*n*: not initialized, can't send message
An `M_DATA` message was passed to the driver for a channel that had not been programmed at least once since the driver was loaded. The ESCC's registers were in an unknown state. The `S_IOCSETMODE` ioctl command performs the programming operation.

se*n* hdlc_start: Invalid message type *d* on write queue
driver received an invalid message type from streams.

se_hdlc*n*: transmit hung
The transmitter was not successfully restarted after the watchdog timer expired. This is usually caused by a bad or disconnected cable.

Name ses – SCSI enclosure services device driver

Synopsis `ses@`*target*`,`*lun*

Description The `ses` device driver is an interface to SCSI enclosure services devices. These devices sense and monitor the physical conditions in an enclosure as well as allow access to the status reporting and configuration features of the enclosure (such as indicator LEDs on the enclosure.)

`ioctl(9E)` calls may be issued to `ses` to determine the state of the enclosure and to set parameters on the enclosure services device.

No `ses` driver properties are defined. Use the `ses.conf` file to configure the `ses` driver.

Examples **EXAMPLE 1** `ses.conf` File Format

The following is an example of the `ses.conf` file format:

```
#
# Copyright (c) 1996, by Sun Microsystems, Inc.
# All rights reserved.
#
#
#ident "@(#)ses.conf 1.1    97/02/10 SMI"
#

name="ses" parent="sf"
      target=15;

name="ses" parent="SUNW,pln" port=0 target=15;
name="ses" parent="SUNW,pln" port=1 target=15;
name="ses" parent="SUNW,pln" port=2 target=15;
name="ses" parent="SUNW,pln" port=3 target=15;
name="ses" parent="SUNW,pln" port=4 target=15;
name="ses" parent="SUNW,pln" port=5 target=15;

name="ses" class="scsi"
      target=15 lun=0;
```

ioctls The SES driver currently supports the SES, SAFTE and SEN enclosure service chipsets. SEN and SAFTE protocols are translated internally in the driver into SES compliant data structures. This enables the SES driver to work seamlessly with different protocols and eliminates the need to enhance user applications.

`SESIOC_GETNOBJ` Returns an unsigned integer that represents the number of SES data structures in the enclosure services chip.

`SESIOC_GETOBJMAP` Returns a size array containing `ses_object` elements communicated through `SESIOC_GETNOBJ()`. `ses_object` is defined in `sesio.h`.

`SESIOC_INIT`	Instructs the device to perform a self-diagnostic test. Currently SES & SEN devices always return success.
`SESIOC_GETENCSTAT`	Returns an unsigned character that represents status enclosure as defined by Table 25 in Section 7.1.2 of the SES specification *NCITS 305-199x*.
`SESIOC_GETOBJSTAT`	This ioctl is passed an `ses_objarg` containing the `obj_id` you want to set, then fills in the remaining fields according to element status page of the SES specification.
`SESIOC_SETOBJSTAT`	Sets options in the control field. You set control field options by filling out all fields in `ses_objarg`. Field definitions are presented in Section 7.2.2 of the SES specification.

Files `/kernel/drv/ses.conf` Driver configuration file

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC

See Also `driver.conf(4)`, `scsi(4)`, `attributes(5)`, `esp(7D)`, `isp(7D)`, `ioctl(9E)`

Name sesio – enclosure services device driver interface

Synopsis

```
#include <sys/sesio.h>
```

Description The `ses` device driver provides the following ioctls as a means to access SCSI enclosure services devices.

ioctls The `ses` driver supports the following ioctls:

`SES_IOCTL_GETSTATE` This ioctl obtains enclosure state in the `ses_ioctl` structure.

`SES_IOCTL_SETSTATE` This ioctl is used to set parameters on the enclosure services device. The `ses_ioctl` structure is used to pass information into the driver.

Errors

`EIO` The `ses` driver was unable to obtain data from the enclosure services device or the data transfer could not be completed.

`ENOTTY` The `ses` driver does not support the requested ioctl function.

`ENXIO` The enclosure services device does not exist.

`EFAULT` The user specified a bad data length.

Structures The `ses_ioctl` structure has the following fields:

```
uint32_t;              /* Size of buffer that follows */
uint8_t page_code:     /* Page to be read/written */
uint8_t reserved[3];   /* Reserved; Set to 0 */
unit8t  buffer[1];     /* Size arbitrary, user specifies */
```

Examples **EXAMPLE 1** Using the `SES_IOCTL_GETSTATE` ioctl

The following example uses the `SES_IOCTL_GETSTATE` ioctl to recover 20 bytes of page 4 from a previously opened device.

```
char   abuf[30];
struct ses_ioctl *sesp;
int    status;
sesp = (ses_ioctl *)abuf;
sesp->size = 20;
sesp->page_code = 4;
status = ioctl(fd, SES_IOCTL_GETSTATE, abuf);
```

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC

See Also ses(7D), ioctl(9E)

Name sf – SOC+ FC-AL FCP Driver

Synopsis `sf@`*port*`,0`

Description The `sf` driver is a SCSA compliant nexus driver which supports the Fibre Channel Protocol for SCSI on Private Fibre Channel Arbitrated loops. An SBus card called the SOC+ card (see `socal(7D)`) connects the Fibre Channel loop to the host system.

The `sf` driver interfaces with the SOC+ device driver, `socal(7D)`, the SCSI disk target driver, `ssd(7D)`, and the SCSI-3 Enclosure Services driver, `ses(7D)`. It only supports SCSI devices of type disk and ses.

The `sf` driver supports the standard functions provided by the SCSA interface. The driver supports auto request sense and tagged queueing by default.

The driver requires that all devices have unique hard addresses defined by switch settings in hardware. Devices with conflicting hard addresses will not be accessible.

Files

/platform/*architecture*/kernel/drv/sf	ELF kernel module
/platform/*architecture*/kernel/drv/sf.conf	`sf` driver configuration file

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC

See Also `luxadm(1M)`, `prtconf(1M)`, `driver.conf(4)`, `socal(7D)`, `ssd(7D)`

Writing Device Drivers

ANSI X3.272-1996, Fibre Channel Arbitrated Loop (FC-AL)

ANSI X3.269-1996, Fibre Channel Protocol for SCSI (FCP)

ANSI X3.270-1996, SCSI-3 Architecture Model (SAM)

Fibre Channel Private Loop SCSI Direct Attach (FC-PLDA)

Diagnostics In addition to being logged, the messages below may display on the system console.

The first set of messages indicate that the attachment was unsuccessful, and will only display while the `sf` driver is initially attempting to attach. Each message is preceded by `sf`*%d* , where *%d* is the instance number of the `sf` device.

Failed to alloc soft state	Driver was unable to allocate space for the internal state structure. Driver did not attach to device, SCSI devices will be inaccessible.

Bad soft state	Driver requested an invalid internal state structure. Driver did not attach to device, SCSI devices will be inaccessible.
Failed to obtain transport handle	Driver was unable to obtain a transport handle to communicate with the socal driver. Driver did not attach to device, SCSI devices will be inaccessible
Failed to allocate command/response pool	Driver was unable to allocate space for commands and responses. Driver did not attach to device, SCSI devices will be inaccessible.
Failed to allocate kmem cache	Driver was unable to allocate space for the packet cache. Driver did not attach to device, SCSI devices will be inaccessible.
Failed to allocate dma handle for	Driver was unable to allocate a dma handle for the loop map. Driver did not attach to device, SCSI devices will be inaccessible.
Failed to allocate lilp map	Driver was unable to allocate space for the loop map. Driver did not attach to device, SCSI devices will be inaccessible.
Failed to bind dma handle for	Driver was unable to bind a dma handle for the loop map. Driver did not attach to device, SCSI devices will be inaccessible.
Failed to attach	Driver was unable to attach for some reason that may be printed. Driver did not attach to device, SCSI devices will be inaccessible.

The next set of messages may display at any time. The full device pathname, followed by the shorter form described above, will precede the message.

Invalid lilp map	The driver did not obtain a valid lilp map from the socal driver. SCSI device will be inaccessible.
Target t, AL-PA x and hard	The device with a switch setting t has an AL-PA x which does not match its hard address y. The device will not be accessible.
Duplicate switch settings	The driver detected devices with the same switch setting. All such devices will be inaccessible.
WWN changed on target t	The World Wide Name (WWN) has changed on the device with switch setting t.

Target t, unknown device type	The driver does not know the device type reported by the device with switch setting t.

Name sfe – SiS900 series Fast Ethernet device driver

Synopsis `/dev/sfe`

Description The `sfe` driver is a loadable, clonable, GLD-based STREAMS driver supporting the Data Link Provider Interface `dlpi(7P)` on Silicon Integrated Systems 900 series Fast Ethernet controllers.

Application Programming Interface The relevant fields returned as part of a `DL_INFO_ACK` response are:

- Maximum SDU is 1500.
- Minimum SDU is 0.
- The dlsap address length is 8.
- MAC type is `DL_ETHER`.
- SAP length is -2. The 6–byte physical address is followed immediately by a 2–byte SAP.
- Service mode is `DL_CLDLS`.
- Broadcast address is the 6-byte Ethernet broadcast address (`ff:ff:ff:ff:ff:ff`).

If the SAP provided is zero, then *IEEE 802.3* mode is assumed and outbound frames will have the frame payload length written into the type field. Likewise, inbound frames with a SAP between zero and 1500 are interpreted as *IEEE 802.3* frames and delivered to any streams that are bound to SAP zero (the *802.3* SAP).

Properties The following properties may be configured using either `ndd(1M)` or the `sfe.conf` configuration file as described by `driver.conf(4)`:

`adv_100fdx_cap`
Enables the 100 Base TX full-duplex link option. (This is generally the fastest mode if both link partners support it. Most modern equipment supports this mode.)

`adv_100hdx_cap`
Enables the 100 Base TX half-duplex link option. (Typically used when the link partner is a 100 Mbps hub.)

`adv_10fdx_cap`
Enables the 10 Base-T full-duplex link option. (This less-frequently used mode is typically used when the link partner is a 10 Mbps switch.)

`adv_10hdx_cap`
Enables the 10 Base-T half-duplex link option. (This is the fall-back when no other option is available. It is typically used when the link partner is a 10 Mbps hub or is an older network card.)

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SPARC, x86
Interface Stability	Committed

Files

`/dev/sfe`
Special character device.

`/kernel/drv/sfe`
32-bit driver binary (x86).

`/kernel/drv/amd64/sfe`
64-bit driver binary (x86).

`/kernel/drv/sparcv9/sfe`
64-bit driver binary (SPARC).

`/kernel/drv/sfe.conf`
Configuration file.

See Also `ndd(1M)`, `driver.conf(4)`, `attributes(5)`, `streamio(7I)`, `dlpi(7P)`

IEEE 802.3 — Institute of Electrical and Electronics Engineers, 2002

Name sgen – Generic SCSI device driver

Synopsis

```
#include <sys/scsi/targets/sgendef.h>

sgen@target,lun:<devtype>
```

Description The sgen driver exports the uscsi(7I) interfaces to user processes. The sgen driver can be configured to bind to SCSI devices for which no system driver is available. Examples of such devices include SCSI scanners and SCSI processor devices.

Security Typically, drivers which export the uscsi(7I) interface unconditionally require that the user present superuser credentials. The sgen driver does not, and relies on the filesystem permissions on its device special file to govern who may access that device. By default, access is restricted and device nodes created by the sgen driver are readable and writable by the superuser exclusively.

It is important to understand that SCSI devices coexisting on the same SCSI bus may potentially interact with each other. This may result from firmware bugs in SCSI devices, or may be made to happen programmatically by sending appropriate SCSI commands to a device. Potentially, any application controlling a device via the sgen driver can introduce data integrity or security problems in that device or any other device sharing the same SCSI bus.

Granting unprivileged users access to an sgen-controlled SCSI device may create other problems. It may be possible for a user to instruct a target device to gather data from another target device on the same bus. It may also be possible for malicious users to install new firmware onto a device to which they are granted access. In environments where security is a concern but user access to devices controlled by the sgen driver is nontheless desired, it is recommended that the devices be separated onto a dedicated SCSI bus to mitigate the risk of data corruption and security violations.

Configuration The sgen driver is configurable via the `sgen.conf` file. In addition to standard SCSI device configuration directives (see scsi(4)), administrators can set several additional properties for the sgen driver.

By default, the sgen driver will not claim or bind to any devices on the system. To do so, it must be configured by the administrator using the `inquiry-config-list` and/or the `device-type-config-list` properties.

As with other SCSI drivers, the `sgen.conf` configuration file enumerates the targets sgen should use. See scsi(4) for more details. For each target enumerated in the `sgen.conf` file, the sgen driver sends a `SCSI INQUIRY` command to gather information about the device present at that target. The `inquiry-config-list` property specifies that the sgen driver should bind to a particular device returning a particular set of inquiry data. The `device-type-config-list` specifies that the sgen driver should bind to every device that is of a particular SCSI device type. When examining the device, the sgen driver tests to see if it matches an entry in the `device-type-config-list` or the `inquiry-config-list`. For more detail on these two properties, see the PROPERTIES section.

When a match against the INQUIRY data presented by a device is made, the sgen driver attaches to that device and creates a device node and link in the /devices and /dev hierarchies. See the FILES section for more information about how these files are named.

It is important for the administrator to ensure that devices claimed by the sgen driver do not conflict with existing target drivers on the system. For example, if the sgen driver is configured to bind to a direct access device, the standard sd.conf file will usually cause sd to claim the device as well. This can cause unpredictable results. In general, the uscsi(7I) interface exported by sd(7D) or st(7D) should be used to gain access to direct access and sequential devices.

The sgen driver is disabled by default. The sgen.conf file is shipped with all of the 'name="sgen" class="scsi" target=...' entries commented out to shorten boot time and to prevent the driver from consuming kernel resources. To use the sgen driver effectively on desktop systems, simply uncomment all of the name="sgen" lines in sgen.conf file. On larger systems with many SCSI controllers, carefully edit the sgen.conf file so that sgen binds only where needed. Refer to driver.conf(4) for further details.

Properties

inquiry-config-list — The inquiry-config-list property is a list of pairs of strings that enumerates a list of specific devices to which the sgen driver will bind. Each pair of strings is referred to as <vendorid, productid> in the discussion below.

vendorid — is used to match the Vendor ID reported by the device. The SCSI specification limits Vendor IDs to eight characters. Correspondingly, the length of this string should not exceed eight characters. As a special case, "*" may be used as a wildcard which matches any Vendor ID. This is useful in situations where more than one vendor produces a particular model of a product. vendorid is matched against the Vendor ID reported by the device in a case-insensitive manner.

productid — is used to match the product ID reported by the device. The SCSI specification limits product IDs to sixteen characters (unused characters are filled with the whitespace characters). Correspondingly, the length of productid should not exceed sixteen characters. When examining the product ID of the device, sgen examines the length l of productid and performs a match against only the first l characters in the device's product ID. productid is matched against the product ID reported by the device in a case-insensitive manner.

For example, to match some fictitious devices from ACME corp, the inquiry-config-list can be configured as follows:

```
inquiry-config-list =           "ACME",          "UltraToast 3000",
                                "ACME",          "UltraToast 4000",
                                "ACME",          "UltraToast 5000";
```

To match "UltraToast 4000" devices, regardless of vendor, `inquiry-config-list` is modified as follows:

```
inquiry-config-list =           "*",              "UltraToast 4000";
```

To match every device from ACME in the "UltraToast" series (i.e UltraToast 3000, 4000, 5000, ...), `inquiry-config-list` is modified as follows:

```
inquiry-config-list =            "ACME"          "UltraToast";
```

Whitespace characters *are* significant when specifying `productid`. For example, a `productid` of "UltraToast 1000" is fifteen characters in length. If a device reported its ID as "UltraToast 10000", the `sgen` driver would bind to it because only the first fifteen characters are considered significant when matching. To remedy this situation, specify `productid` as "UltraToast 1000 ", (note trailing space). This forces the `sgen` driver to consider all sixteen characters in the product ID to be significant.

`device-type-config-list` The `device-type-config-list` property is a list of strings that enumerate a list of device types to which the `sgen` driver will bind. The valid device types correspond to those defined by the *SCSI-3 SPC Draft Standard, Rev. 11a*. These types are:

Type Name	Inquiry Type ID
direct	0x00
sequential	0x01
printer	0x02
processor	0x03
worm	0x04
rodirect	0x05
scanner	0x06
optical	0x07
changer	0x08
comm	0x09
prepress1	0x0a
prepress2	0x0b

Type Name	Inquiry Type ID
array_ctrl	0x0c
ses	0x0d
rbc	0x0e
ocrw	0x0f
bridge	0x10
type_unknown	0x1f

Alternately, you can specify device types by `INQUIRY` type ID. To do this, specify `type_0x<typenum>` in the `sgen-config-list`. Case is not significant when specifying device type names.

`sgen-diag` The `sgen-diag` property sets the diagnostic output level. This property can be set globally and/or per target/lun pair. `sgen-diag` is an integer property, and can be set to 0, 1, 2 or 3. Illegal values will silently default to 0. The meaning of each diagnostic level is as follows:

`0` No error reporting [default]

`1` Report driver configuration information, unusual conditions, and indicate when sense data has been returned from the device.

`2` Trace the entry into and exit from routines inside the driver, and provide extended diagnostic data. No error reporting [default].

`3` Provide detailed output about command characteristics, driver state, and the contents of each CDB passed to the driver.

In ascending order, each level includes the diagnostics that the previous level reports. See the IOCTLS section for more infomation on the `SGEN_IOC_DIAG` ioctl.

Files

`sgen.conf` Driver configuration file. See CONFIGURATION for more details.

`/dev/scsi/<devtype>/c`*n*`t`*n*`d`*n* The `sgen` driver categorizes each device in a separate directory by its SCSI device type. The files inside the directory are named according to their controller number, target ID and LUN as follows:

c*n* is the controller number, t*n* is the SCSI target id and d*n* is the SCSI LUN

This is analogous to the {`controller;target;device`} naming scheme, and the controller numbers correspond to the same controller numbers which are used for

naming disks. For example, `/dev/dsk/c0t0d0s0` and `/dev/scsi/scanner/c0t5d0` are both connected to controller `c0`.

ioctls The `sgen` driver exports the `uscsi(7I)` interface for each device it manages. This allows a user process to talk directly to a SCSI device for which there is no other driver installed in the system. Additionally, the `sgen` driver supports the following ioctls:

`SGEN_IOC_READY` Send a `TEST UNIT READY` command to the device and return 0 upon success, non-zero upon failure. This ioctl accepts no arguments.

`SGEN_IOC_DIAG` Change the level of diagnostic reporting provided by the driver. This ioctl accepts a single integer argument between 0 and 3. The levels have the same meaning as in the `sgen-diag` property discussed in PROPERTIES above.

Errors

`EBUSY` The device was opened by another thread or process using the O_EXCL flag, or the device is currently open and O_EXCL is being requested.

`ENXIO` During opening, the device did not respond to a `TEST UNIT READY` SCSI command.

`ENOTTY` Indicates that the device does not support the requested ioctl function.

Examples Here is an example of how `sgen` can be configured to bind to scanner devices on the system:

```
device-type-config-list = "scanner";
```

The administrator should subsequently uncomment the appropriate `name="sgen"...` lines for the SCSI target ID to which the scanner corresponds. In this example, the scanner is at target 4.

```
name= "sgen" class= "scsi" target=4 lun=0;
```

If it is expected that the scanner will be moved from target to target over time, or that more scanners might be added in the future, it is recommended that all of the `name="sgen"...` lines be uncommented, so that `sgen` checks all of the targets on the bus.

For large systems where boot times are a concern, it is recommended that the `parent=""` property be used to specify which SCSI bus `sgen` should examine.

See Also `driver.conf(4)`, `scsi(4)`, `sd(7D)`, `st(7D)`, `uscsi(7I)`

Writing Device Drivers

ANSI Small Computer System Interface-2 (SCSI-2)

SCSI-3 SPC Draft Standard, Rev. 11a

Name sharefs – Kernel sharetab filesystem

Description The `sharefs` filesystem describes the state of all shares currently loaded by the kernel. It is mounted during boot time as a read-only file at `/etc/dfs/sharetab`.

Filesystem contents are dynamic and reflect the current set of shares in the system. File contents are described in `sharetab(4)`.

File contents can be modified as a result of `share(1M)`, `sharectl(1M)`, `sharemgr(1M)` and changing properties of a `zfs(1M)` data set.

The module may not be unloaded dynamically by the kernel.

Files `/etc/dfs/sharetab` System record of shared file systems.

See Also `share(1M)`, `sharectl(1M)`, `sharemgr(1M)`, `zfs(1M)`, `sharetab(4)`

Name si3124 – SiliconImage 3124/3132 SATA controller driver

Synopsis `sata@unit-address`

Description The `si3124` driver is a SATA framework-compliant HBA driver that supports Silicon Image 3124 and 3132 SATA controllers. Note that while the Silicon Image controllers supports standard SATA features including SATA-II disks, NCQ, hotplug, port multiplier and ATAPI disks, the `si3124` driver currently does not support NCQ, port multiplier or ATAPI features.

Configuration There are no tunable parameters in the `si3124.conf` file.

Files

`/kernel/drv/si3124`	32–bit ELF kernel module (x86).
`/kernel/drv/amd64/si3124`	64–bit ELF kernel module. (x86).

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWsi3124

See Also `cfgadm(1M)`, `prtconf(1M)`, `cfgadm_sata(1M)`, `attributes(5)`, `nv_sata(7D)`, `sata(7D)`

Writing Device Drivers

Name sip – SIP Proxy/registrar/redirect server

Description Solaris supports deployment of VoIP/SIP services by providing an *RFC 3261*–compliant SIP proxy/registrar/redirect server called SER from *iptel.org*.

See the `ser(8)` man page under `/usr/sfw/man`.

Files `/etc/sfw/ser/ser.cfg`

`/etc/sfw/ser/README.solaris.ser`

`/usr/sfw/share/doc/ser/README`

Name sk98sol – SysKonnect Gigabit Ethernet SK-98xx device driver

Synopsis

```
/dev/skge

/kernel/drv/sk98sol
```

Description The `sk98sol` driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface (DLPI), over a SysKonnect Gigabit Ethernet adapter (SK-98*xx* series). The driver supports multiple installed SysKonnect SK-98*xx* adapters. Functions include chip initialization, frame transmit and receive, multicast and promiscuous support, and error recovery and reporting.

Application Programming Interface The driver provides the `/dev/skge` cloning character-special device as well as per-adapter character-special devices `/dev/skge`*x*, where *x* represents the device instance number.

sk98sol and DLPI

The `sk98sol` driver is a Style 1 and Style 2 Data Link Service (DLS) provider. All `M_PROTO` and `M_PCPROTO` type messages are interpreted as DLPI primitives. Valid DLPI primitives are defined in `<sys/dlpi.h>`. See `dlpi(7P)`.

An explicit `DL_ATTACH_REQ` message by the user is required to associate the opened stream with a particular device (*ppa*). This is unnecessary and invalid for DLPI Style 1. The *ppa* ID is interpreted as an `unsigned long` data type and indicates the corresponding device instance (unit) number. An error (`DL_ERROR_ACK`) is returned by the driver if the *ppa* field value does not correspond to a valid device instance number for the system.

The device is initialized on first attach and de-initialized (stopped) upon last detach. Valid device numbers for all detected adapters are displayed on the console at driver startup time and are written to the `/var/adm/messages` log file.

The values returned in the `DL_INFO_ACK` primitive in response to the `DL_INFO_REQ` request are:

- Maximum SDU is `1500` (`9000` if *JumboFrames* are enabled).
- Minimum SDU is `0`.
- DLSAP address length is `8` bytes.
- MAC type is `DL_CSMACD`.
- SAP length value is –2, meaning the physical address component is followed immediately by a 2-byte SAP component within the DLSAP address.
- Service mode is `DL_CLDLS`.
- Optional quality of service (QOS) support is not included; as a result, the QOS field values are `0`.
- Provider style is `DL_STYLE2`.
- Version is `DL_VERSION_2`.

Options Parameters are set in the `/kernel/drv/sk98sol.conf` configuration file, which is created during installation. See `driver.conf(4)`. You can edit the `/kernel/drv/sk98sol.conf` file to reflect your settings and reboot the system to use the new parameter values. If the file exists prior to driver installation, the new parameter values will be used as soon as the driver is installed.

String parameter values must be surrounded with double quotes ("), while integer parameter values are not. Parameter names and values are case sensitive and you should use them exactly as shown.

Note – You can increase `sk98sol` performance by tuning certain TCP and UDP parameters. However, you should be aware that this act may adversely impact the performance of other network cards.

To tune specific parameters to increase `sk98sol` performance, do the following:

```
ndd -set /dev/tcp tcp_xmit_hiwat      65536
ndd -set /dev/tcp tcp_xmit_lowat      32768
ndd -set /dev/tcp tcp_recv_hiwat      65536

ndd -set /dev/udp udp_xmit_hiwat      65536
ndd -set /dev/udp udp_xmit_lowat      327
ndd -set /dev/udp udp_recv_hiwat      65536
```

Per-Port Parameters The parameters discussed in this section can be set for each port on the adapter.

In each of the following descriptions, *?* represents port A or B.

AutoNegotiation_?

Type:	`String`
Values:	`On`, `Off`, `Sense`
Default:	`Sense` (for SK-984*x* fiber adapters).
Default:	`On` (for SK-982*x* copper adapters.)

The `Sense` value automatically detects whether the link partner supports autonegotiation. If your link partner is configured to half duplex with autonegotiation turned off, set the *AutoNegotiation_?* and *DuplexCapabilities_?* parameters manually. Do *not* set the *AutoNegotiation_?* parameter value to `Sense`, as it will fail.

Do not use `Sense` for 1000Base-T (copper) adapters. If `Sense` is selected, it will be mapped to `On` automatically.

DuplexCapabilities_?

Type:	`String`
Values:	`Half`, `Full`, `Both`
Default:	`Both`

Set the *DuplexCapabilities_?* parameter only if the *AutoNegotiation_?* parameter is set to the `On` or `Off` values. If *AutoNegotiation_?* is set to `On`, all three *AutoNegotiation_?* values are possible; however, if set to `Off`, only the `Full` and `Half` values are allowed.

Use the *DuplexCapabilities_?* parameter if your link partner does not support all possible combinations.

FlowControl_?

Type:	`String`
Values:	`Sym`, `SymOrRem`, `LocSend`, None
Default:	`SymOrRem`

Use the *FlowControl_?* parameter to set the flow control capabilities reported by the port during autonegotiation:

`Sym`	Symetric flow control, where both link partners are allowed to send PAUSE frames.
`SymOrRem`	SymetricOrRemote flow control, where both link partners or only the remote partner are allowed to send PAUSE frames.
`LocSend`	LocalSend flow control, where only the local link partner is allowed to send PAUSE frames.
`None`	No flow control, where no link partner is allowed to send PAUSE frames.

The *FlowControl_?* parameter is ignored if *AutoNegotiation_?* is set to "`Off.`"

Role_?

Type:	`String`
Values:	`Auto`, `Master`, `Slave`
Default:	`Auto`

Use the *Role_?* parameter only for the SK-9821 and SK-9822 adapters.

1000Base-T communication between two ports requires one port to act as the master (and provide timing information) and the other as slave. Normally, this is negotiated between the two ports during link establishment. If this fails, use the *Role_?* parameter to force the master and slave roles on the ports. If *AutoNegotiation_?* is set to "`Off`," then the *Role_?* parameter must be set manually.

Per-Adapter Parameters

PreferredPort

Type:	`String`
Values:	`A`, `B`
Default:	`A`

Use the *PreferredPort* parameter to force the preferred port to `A` or `B` (on two-port NICs). The preferred port is the port selected if both ports are detected as fully functional.

RlmtMode

Type:	`integer`
Values:	`1`, `2`, `3`
Default:	`1`

RLMT (Redundant Link Management Technology) provides three modes to determine if a port is available for use.

1. Check link state only: use the link state reported by the adapter hardware for each individual port.
2. Check other port: RLMT sends test frames from one port to another and checks if they are received. The ports must be connected to the network that allow LLC test frames to be exchanged (that is, networks without routers between the ports).
3. Check other port and segmentation: RLMT checks the other port and also requests information from the Gigabit Ethernet switch next to each port to determine if the network is segmented between the ports. Only use this mode if you have Gigabit Ethernet switches installed and configured to use the Spanning Tree protocol.

Note that modes 2 and 3 are meant to operate in configurations where a network path exists between the ports on a single adapter. They are *not* designed to work in networks where adapters are connected back-to-back.

JumboFrames

Type:	`String`
Values:	`Off`, `On`
Default:	`Off`

To enable support for *JumboFrames* (frames with a length of up to 9014 bytes), set `JumboFrames` to "`On`." Because longer frames reduce operating system overhead, *JumboFrames* increases network throughput.

For full *JumboFrames* support, the maximum transfer unit (MTU) size used by TCP/IP must also be changed by using the `ifconfig(1M)` command. To do this, remove the comment sign (`#`) before the `ifconfig` line in the `/etc/rcS.d/S50sk98sol` file. You should also ensure that the adapter device number (`skge0`) matches the attach number displayed during system startup. The MTU must be set to 9000, not including the 14 bytes of MAC address header.

JumboFrames can only be used if *all* equipment in your subnetwork supports them; currently many switches do not support *JumboFrames*). Devices without Jumbo Support drop the longer frames (and might report them as error frames). If you experience problems with this, connect two SK-98*xx* adapters (with *JumboFrames* enabled) back-to-back.)

CopyThreshold

Type:	`Integer`
Values:	`0`–`1500`
Default:	`1500`

During transmit, the driver relies on the frame's physical memory address to tell the hardware where to find the frame data. Setting up the DMA address can take time on Solaris; it may be more convenient to copy the frame data to a buffer that you have set up in advance. All frames with a length less than or equal to the *CopyThreshold* parameter value are copied into buffers; for longer frames, the real DMA setup is done. By default (without *JumboFrames* support), all frames are copied. You can experiment with this parameter to find out if your system performs better with only smaller frames copied.

To use more complex syntax for setting different parameters on multiple adapters, see `driver.conf(4)`. For example:

```
name="sk98sol" parent="/pci@1f,4000" unit-address="2" AutoNegotiation_A="Off";
name="sk98sol" parent="/pci@1f,2000" unit-address="2" AutoNegotiation_B="Sense";
```

Diagnostics If multiple NICs are installed in the system, the following message may appear on the console and in the `/var/adm/messages` log file:

```
Allocation of descriptor memory failed
```

You can avoid this message by tuning the `lomempages` kernel parameter. By default, the value of this parameter is 36 pages. Each SK-98*xx* adapter requires a determined number of pages, so increase the value of the `lomempages` parameter in increments of ten pages until all NICs in the system run correctly.

To modify the value of this parameter to 46 pages, append the `set lomempages=46` line to the `/etc/system` file and reboot the system.

Files

`/dev/skge`	Character special device
`/dev/skge`*x*	Per-adapter character special device, where *x* is the adapter *ppa*
`/kernel/drv/sk98sol`	ELF kernel module
`/kernel/drv/sparcv9/sk98sol`	64–bit ELF kernel module
`/kernel/drv/sk98sol.conf`	Driver configuration file

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	IA, SPARC

See Also `ifconfig(1M)`, `netstat(1M)`, `driver.conf(4)`, `attributes(5)`, `dlpi(7P)`.

`sk98sol.txt` driver README file — Included in the driver package; also available from *www.syskonnect.com*.

Name skfp – SysKonnect FDDI PCI device driver

Synopsis `/dev/skfp`

Description The `skfp` FDDI driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface (DLPI) over a SysKonnect FDDI PCI adapter. The driver supports multiple installed SysKonnect FDDI PCI adapters. Functions include chip initialization, frame transit and receive, multicast and promiscuous support, and error recovery and reporting.

The `skfp` driver supports all SysKonnect SK-NET FDDI PCI adapters (SK-55*xx* (32-bit) and SK-58*xx* (64-bit) series) on 32-bit systems, and the SK-58*xx* series on 64-bit systems.

Application Programming Interface The `skfp` driver provides the `/dev/skfp` cloning character-special device that accesses all SK-NET FDDI PCI adapters using Data Link Service (DLS) Style 2. It also provides per-adapter character-special devices `/dev/skfp`*x*, (where *x* represents the device instance number) that access a special NIC using DLS Style 1.

skfp and DLPI The `skfp` driver is a Style 1 and Style 2 DLS provider. All `M_PROTO` and `M_PCPROTO` type messages are interpreted as DLPI primitives. Valid DLPI primitives are defined in `<sys/dlpi.h>`. See `dlpi(7P)`.

An explicit `DL_ATTACH_REQ` message by the user is required to associate the opened Stream with a particular device (*ppa*). This is unnecessary and invalid for DLPI Style 1. The *ppa* ID is interpreted as an `unsigned long` data type and indicates the corresponding device instance (unit) number. An error (`DL_ERROR_ACK`) is returned by the driver if the *ppa* field value does not correspond to a valid device instance number for the system.

The device is initialized on first attach and de-initialized (stopped) upon last detach. Valid device numbers for all detected adapters are displayed on the console at driver startup time and written to the `/var/adm/messages` log file.

The values returned in the `DL_INFO_ACK` primitive in response to the `DL_INFO_REQ` request are:

- Maximum SDU is `4470`.
- Minimum SDU is `0`.
- DSLAP address length is 8 bytes.
- MAC type is `DL_FDDI`.
- SAP length value is –2, meaning the physical address component is followed immediately by a 2-byte SAP component within the DLSAP address.
- Service mode is `DL_CLDLS`.
- Optional quality of service (QOS) support is not included; as a result, the QOS field values are `0`.
- Provider style is `DL_STYLE2`.
- Version is `DL_VERSION_2`.

Options Options are not required for normal operation. In special cases, FDDI Station Management (SMT) parameters can be modified by using the `/usr/bin/smtpara` utility (see the driver README files). The `smtpara` utility should be used only by those very familiar with FDDI.

DIAGNOSTICS If multiple NICs are installed in the system, the following message may appear on the console and in the `/var/adm/messages` log file:

```
skfp: DMA memory allocation failed !
```

You can avoid this message by tuning the `lomempages` kernel parameter. By default, the value of this parameter is 36 pages. Each SK-FDDI PCI adapter requires nine pages, so increase the value of the `lomempages` parameter in increments of nine pages until all NICs in the system run correctly.

To modify the value of this parameter to 45 pages, you can, for example, append the `set lomempages=45` line to the `/etc/system` file and reboot the system.

Files

`/dev/skfp`	Character special device
`/dev/skfp`*x*	Per-adapter character special device, where *x* is the adapter *ppa*
`/kernel/drv/skfp`	ELF kernel module
`/kernel/drv/skfp.conf`	Driver configuration file
`/usr/bin/smtpara`	SMT parameter utility
`/etc/fddi.cfg`	`smtpara` configuration file
`<sys/dlpi.h>`	DLPI definitions

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	IA, SPARC

See Also `ifconfig(1M)`, `netstat(1M)`, `attributes(5)`, `dlpi(7P)`

`skfp.txt` README file — Included in the driver package or available from *www.syskonnect.com*.

Name slp – Service Location Protocol

Description The Service Location Protocol (SLP) is a dynamic service discovery protocol that runs on top of the Internet Protocol (IP). The protocol is specified by the IETF standard-track documents *RFC 2165*, *RFC 2608*, *RFC 2609*; the API is documented in *RFC 2614*. .

There are two components to the SLP technology. The first is a daemon, `slpd(1M)`, which coordinates SLP operations. The second is a software library, `slp_api(3SLP)`, through which processes access a public API. Both components are configured by means of the SLP configuration file, `slp.conf(4)`.

The SLP API is useful for two types of processes:

Client Applications	Services and service information can be requested from the API. Clients do not need to know the location of a required service, only the type of service, and optionally, the service characteristics. SLP will supply the location and other information to the client through the API.
Server Processes	Programs that offer network services use the SLP API to advertise their location as well as other service information. The advertisement can optionally include attributes describing the service. Advertisements are accompanied by a lifetime; when the lifetime expires, the advertisement is flushed, unless it is refreshed prior to expiration.

API libraries are available for both the C and Java languages.

SLP provides the following additional features:

- `slpd(1M)` can be configured to function as a transparent directory agent. This feature makes SLP scalable to the enterprise. System administrators can configure directory agents to achieve a number of different strategies for scalability.
- SLP service advertising and discovery is performed in scopes. Unless otherwise configured, all discovery and all advertisements are in the scope *default*. In the case of a larger network, scopes can be used to group services and client systems so that users will only find those services which are physically near them, belong to their department, or satisfy the specified criteria. Administrators can configure these scopes to achieve different service provider strategies.
- Services may be registered by proxy through a serialized registration file. This is an alternative to registering services through the API. See `slpd.reg(4)` for more information.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWslpu
CSI	CSI-enabled
Interface Stability	Standard
MT-Level	MT-Safe

See Also `slpd(1M)`, `slp_api(3SLP)`, `slp.conf(4)`, `slpd.reg(4)`, `attributes(5)`

Guttman, E., Perkins, C., Veizades, J., and Day, M., *RFC 2608, Service Location Protocol, Version 2*, The Internet Society, June 1999.

Guttman, E., Perkins, C., and Kempf, J., *RFC 2609, Service Templates and Service: Schemes*, The Internet Society, June 1999.

Kempf, J. and Guttman, E., *RFC 2614, An API for Service Location*, The Internet Society, June 1999.

Veizades, J., Guttman, E., Perkins, C., and Kaplan, S., *RFC 2165, Service Location Protocol*, Network Working Group, 1997.

Name smbfs – CIFS/SMB file system

Description The `smbfs` file system allows you to mount CIFS shares that are exported from Windows or compatible systems. SMB is the historical name for the CIFS protocol, which stands for Server Message Block and is more commonly used in technical contexts.

The `smbfs` file system permits ordinary UNIX applications to change directory into an `smbfs` mount and perform simple file and directory operations. Supported operations include `open`, `close`, `read`, `write`, `rename`, `delete`, `mkdir`, `rmdir` and `ls`.

Limitations Some local UNIX file systems (for example UFS) have features that are not supported by `smbfs`. These include:

- No mapped-file access because `mmap`(2) returns `ENOSYS`.
- Locking is *local only* and is not sent to the server.

The following are limitations in the CIFS protocol:

- `unlink()` or `rename()` of open files returns `EBUSY`.
- `rename()` of extended attribute files returns `EINVAL`.
- Creation of files with any of the following illegal characters returns `EINVAL`: colon (`:`), backslash (`\\`), slash (`/`), asterisk (`*`), question mark (`?`), double quote (`"`), less than (`<`), greater than (`>`), and vertical bar (`|`).
- `chmod` and `chown` settings are silently discarded.
- Links are not supported.
- Symbolic links are not supported.
- `mknod` is not supported. (Only file and directory objects are supported.)

The current `smbfs` implementation does not support multi-user mounts. Instead, each Unix user needs to make their own private mount points.

Currently, all access through an `smbfs` mount point uses the Windows credentials established by the user that ran the `mount` command. Normally, permissions on `smbfs` mount points should be `0700` to prevent Unix users from using each others' Windows credentials. See the `dirperms` option to `mount_smbfs`(1M) for details regarding how to control `smbfs` mount point permissions.

An important implication of this limitation is that system-wide mounts, such as those made using `/etc/vfstab` or automount maps are only useful in cases where access control is not a concern, such as for public read-only resources.

Attributes See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWsmbfsu
Interface Stability	Uncommitted

See Also smbutil(1), mount_smbfs(1M), nsmbrc(4), attributes(5)

Name smbios – System Management BIOS image

Synopsis `/dev/smbios`

Description The `smbios` device is a character special file that provides access to a snapshot of the System Management BIOS (SMBIOS) image exported by the current system. SMBIOS is an industry-standard mechanism that enables low-level system software to export hardware configuration information to higher-level system management software. The SMBIOS data format is defined by the Distributed Management Task Force (DMTF). For more information on SMBIOS and to obtain a copy of the SMBIOS specification and implementation guidelines, refer to http://*www.dmtf.org*.

The SMBIOS image consists of a table of structures, each describing some aspect of the system software or hardware configuration. The content of the image varies widely by platform and BIOS vendor and may not exist on some systems. You can use the `smbios(1M)` utility to inspect the contents of the SMBIOS image and copy it to a file.

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWkvm.i
Interface Stability	Stable

See Also `prtdiag(1M)`, `smbios(1M)`, `attributes(5)`

System Management BIOS Reference Specification, Version 2, Release 4 — 2005

Notes The implementation of a System Management BIOS image is entirely at the discretion of the system and BIOS vendors. Not all systems export a SMBIOS. The SMBIOS structure content varies widely between systems and BIOS vendors and frequently does not comply with the guidelines included in the specification. For example, some structure fields may not be filled in by the BIOS, while others may be filled in with non-conforming values.

Name smbus – System Management Bus controller driver

Description The `smbus` driver is a I2C (Inter IC) nexus driver that allows the system to communicate with various system component chips. SMBus is a two-wire control bus based on the I2C protocol through which systems can communicate with various I2C devices connected to the bus.

The `smbus` driver supports byte and block level transfer based on interrupt and polled mode.

Files `/platform/sun4u/kernel/drv/sparcv9/smbus` 64 bit ELF kernel module

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcarx
Architecture	SPARC

See Also `attributes(5)`

Writing Device Drivers

System Management Bus (SMBus) Specification 2.0 — SBS Implementation Forum

The I2C Bus and How To Use It —Philips Semiconductor Document # 98-8080-575-01

Name socal – Serial Optical Controller for Fibre Channel Arbitrated Loop (SOC+) device driver

Synopsis `socal@`*sbus-slot*`,0`

Description The Fibre Channel Host Bus Adapter is an SBus card which implements two full duplex Fibre Channel interfaces. Each Fibre Channel interface can connect to a Fibre Channel Arbitrated Loop (FC-AL).

The `socal` device driver is a nexus driver and implements portions of the FC-2 and FC-4 layers of FC-AL.

Files `/kernel/drv/socal` ELF Kernel Module

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC

See Also `sbus(4)`, `sf(7D)`, `ssd(7D)`

Writing Device Drivers

ANSI X3.230-1994, Fibre Channel Physical and Signalling Interface (FC-PH)

ANSI X3.272-1996, Fibre Channel Arbitrated Loop (FC-AL)

Fibre Channel Private Loop SCSI Direct Attach (FC-PLDA)

Diagnostics The messages described below may appear on system console in addition to being logged.

On the console, these messages are preceded by:

`socal`*%d*`: port `*%a*

where *%d* is the instance number of the `socal` controller and *%a* is the port on the host adapter.

Fibre Channel Loop is ONLINE	The Fibre Channel loop is now online.
Fibre Channel Loop is OFFLINE	The Fibre Channel loop is now offline.
attach failed: device in slave-only slot.	Move soc+ card to another slot.
attach failed: bad soft state.	Driver did not attach, devices will be inaccessible.
attach failed: unable to alloc xport struct.	Driver did not attach, devices will be inaccessible.

attach failed: unable to map eeprom	Driver was unable to map device memory; check for bad hardware. Driver did not attach to device, devices will be inaccessible.
attach failed: unable to map XRAM	Driver was unable to map device memory; check for bad hardware. Driver did not attach to device, devices will be inaccessible.
attach failed: unable to map registers	Driver was unable to map device registers; check for bad hardware. Driver did not attach to device, devices will be inaccessible.
attach failed: unable to access status register	Driver was unable to map device registers; check for bad hardware. Driver did not attach to device, devices will be inaccessible.
attach failed: unable to install interrupt handler	Driver was not able to add the interrupt routine to the kernel. Driver did not attach to device, devices will be inaccessible.
attach failed: unable to access host adapter XRAM	Driver was unable to access device RAM; check for bad hardware. Driver did not attach to device, devices will be inaccessible.
attach failed: unable to write host adapter XRAM	Driver was unable to write device RAM; check for bad hardware. Driver did not attach to device, devices will be inaccessible.
attach failed: read/write mismatch in XRAM	Driver was unable to verify device RAM; check for bad hardware. Driver did not attach to device, devices will be inaccessible.

Name sockio – ioctls that operate directly on sockets

Synopsis

```
#include <sys/sockio.h>
```

Description The `ioctls` listed in this manual page apply directly to sockets, independent of any underlying protocol. The `setsockopt()` call (see `getsockopt(3SOCKET)`) is the primary method for operating on sockets, rather than on the underlying protocol or network interface. `ioctls` for a specific network interface or protocol are documented in the manual page for that interface or protocol.

`SIOCSPGRP` The argument is a pointer to an `int`. Set the process-group ID that will subsequently receive `SIGIO` or `SIGURG` signals for the socket referred to by the descriptor passed to `ioctl` to the value of that `int`. The argument must be either positive (in which case it must be a process ID) or negative (in which case it must be a process group).

`SIOCGPGRP` The argument is a pointer to an `int`. Set the value of that `int` to the process-group ID that is receiving `SIGIO` or `SIGURG` signals for the socket referred to by the descriptor passed to `ioctl`.

`SIOCCATMARK` The argument is a pointer to an `int`. Set the value of that `int` to `1` if the read pointer for the socket referred to by the descriptor passed to `ioctl` points to a mark in the data stream for an out-of-band message. Set the value of that `int` to `0` if the read pointer for the socket referred to by the descriptor passed to `ioctl` does not point to a mark in the data stream for an out-of-band message.

See Also `ioctl(2)`, `getsockopt(3SOCKET)`

Name sppptun – PPP tunneling pseudo-driver

Synopsis `/dev/sppptun`

Description The `/dev/sppptun` pseudo-driver provides an interface for tunneling PPP sessions. This interface provides PPP over Ethernet (PPPoE) service with Solaris PPP.

Files `/dev/sppptun` Solaris PPP tunneling device driver.

See Also `pppoec`(1M), `pppoed`(1M), `sppptun`(1M)

RFC 2516 — A Method for Transmitting PPP Over Ethernet (PPPoE). Mamakos, et. al. February 1999.

Name spwr – SMC EtherPower II 10/100 (9432) Ethernet device driver

Synopsis `/dev/spwr`

Description The `spwr` Ethernet driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface, `dlpi(7P)`, over SMC EtherPower II 10/100 controllers. Multiple EtherPower II controllers installed within the system are supported by the driver. The `spwr` driver provides basic support for the SMC EtherPower II hardware. Functions include chip initialization, frame transmit and receive, multicast support, and error recovery and reporting.

Application Programming Interface The cloning character-special device `/dev/spwr` is used to access all SMC EtherPower II devices installed within the system.

The `spwr` driver is dependent on `/kernel/misc/gld`, a loadable kernel module that provides the `spwr` driver with the `DLPI` and STREAMS functionality required of a LAN driver. See `gld(7D)` for more details on the primitives supported by the driver.

The values returned by the driver in the `DL_INFO_ACK` primitive in response to the `DL_INFO_REQ` from the user are as follows:

- The maximum SDU is `1500` (`ETHERMTU`).
- The minimum SDU is `0`. The `spwr` driver will pad to the mandatory 60-octet minimum packet size.
- The `DLSAP` address length is 8.
- The MAC type is `DL_ETHER`.
- The SAP length value is –2, meaning the physical address component is followed immediately by a 2-byte SAP component within the `DLSAP` address.
- The broadcast address value is Ethernet/IEEE broadcast address (`FF:FF:FF:FF:FF:FF`).

Files

`/dev/spwr`	Character special device.
`/kernel/drv/spwr.conf`	Driver configuration file

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86

See Also `attributes(5)`, `dlpi(7P)`, `gld(7D)`

Name srpt – SCSI RDMA Protocol Target Driver for Infiniband (IB)

Description The `srpt` kernel pseudo device driver is an IB Architecture-compliant implementation of the target side of the SCSI RDMA Protocol (SRP). SRP accelerates the SCSI protocol by mapping SCSI data transfer phases to RDMA operations using InfiniBand as the underlying transport.

SRP target services are enabled and disabled through `smf(5)`, using the FMRI `svc:/system/ibsrp/target`.

When enabled, `srpt` enumerates each IB Host Channel Adapter (HCA) on the system and registers each one as a SCSI target using the SCSI Target Mode Framework (STMF).

Files

`/kernel/drv/srpt`	32-bit x86 ELF kernel module
`/kernel/drv/amd64/srpt`	64-bit x86 ELF kernel module
`/kernel/drv/sparcv9/srpt`	64-bit SPARC ELF kernel module
`/kernel/drv/srpt.conf`	Driver configuration file

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWsrptr

See Also `stmfadm(1M)`, `ib(7D)`, `ibdma(7D)`, `ibtl(7D)`, `attributes(5)`

COMSTAR Administration Guide

SCSI RDMA Protocol (SRP) T10 Project 1415-D, Revision

Name ssd – Fibre Channel Arbitrated Loop disk device driver

Synopsis `ssd@`*port*`,`*target*`:`*partition*

Description The `ssd` driver supports Fibre Channel disk devices.

The specific type of each disk is determined by the SCSI inquiry command and reading the volume label stored on block 0 of the drive. The volume label describes the disk geometry and partitioning; it must be present or the disk cannot be mounted by the system.

The block-files access the disk using the system's normal buffering mechanism and are read and written without regard to physical disk records. A "raw" interface provides for direct transmission between the disk and the read or write buffer. A single read or write call usually results in one I/O operation; raw I/O is therefore more efficient when many bytes are transmitted. Block file names are found in `/dev/dsk`; the names of the raw files are found in `/dev/rdsk`.

I/O requests (such as `lseek(2)`) to the SCSI disk must have an offset that is a multiple of 512 bytes (`DEV_BSIZE`), or the driver returns an `EINVAL` error. If the transfer length is not a multiple of 512 bytes, the transfer count is rounded up by the driver.

Partition 0 is normally used for the root file system on a disk, with partition 1 as a paging area (for example, `swap`). Partition 2 is used to back up the entire disk. Partition 2 normally maps the entire disk and may also be used as the mount point for secondary disks in the system. The rest of the disk is normally partition 6. For the primary disk, the user file system is located here.

The device has associated error statistics. These must include counters for hard errors, soft errors and transport errors. Other data may be implemented as required.

Device Statistics Support The device maintains I/O statistics for the device and for partitions allocated for that device. For each device/partition, the driver accumulates reads, writes, bytes read, and bytes written. The driver also initiates hi-resolution time stamps at queue entry and exit points to enable monitoring of residence time and cumulative residence-length product for each queue.

Not all device drivers make per-partition IO statistics available for reporting. `ssd` and `sd(7D)` per-partition statistics are enabled by default but may be disabled in their configuration files.

ioctls Refer to `dkio(7I)`.

Errors

`EACCES`	Permission denied.
`EBUSY`	The partition was opened exclusively by another thread.
`EFAULT`	The argument was a bad address.
`EINVAL`	Invalid argument.
`EIO`	An I/O error occurred.
`ENOTTY`	The device does not support the requested ioctl function.

ENXIO When returned during open(2), this error indicates the device does not exist.

EROFS The device is a read-only device.

Configuration You configure the ssd driver by defining properties in the ssd.conf file. The ssd driver supports the following properties:

enable-partition-kstats The default value is 1, which causes partition IO statistics to be maintained. Set this value to zero to prevent the driver from recording partition statistics. This slightly reduces the CPU overhead for IO, mimimizes the amount of sar(1) data collected and makes these statistics unavailable for reporting by iostat(1M) even though the -p/-P option is specified. Regardless of this setting, disk IO statistics are always maintained.

In addition to the above properties, some device-specific tunables can be configured in ssd.conf using the 'ssd-config-list' global property. The value of this property is a list of duplets. The formal syntax is:

```
ssd-config-list =     <duplet> [, <duplet> ]* ;

where

<duplet>:=        "<vid+pid>" , "<tunable-list>"

and

<tunable-list>:=   <tunable>  [, <tunable> ]*;
<tunable> =        <name> : <value>

The <vid+pid> is the string that is returned by the target device
on a SCSI inquiry command.

The <tunable-list> contains one or more tunables to apply to
all target devices with the specified <vid+pid>.

Each <tunable> is a <name> : <value> pair. Supported
tunable names are:

   delay-busy:    when busy, nsecs of delay before retry.

   retries-timeout:    retries to perform on an IO timeout.
```

Examples The following is an example of a global ssd-config-list property:

```
   ssd-config-list =
      "SUN     T4", "delay-busy:600, retries-timeout:6",
```

```
"SUN     StorEdge_3510", "retries-timeout:3";
```

Files

ssd.conf	Driver configuration file
/dev/dsk/c*n*t*n*d*n*s*n*	block files
/dev/rdsk/c*n*t*n*d*n*s*n*	raw files
c*n*	is the controller number on the system.
t*n*	7-bit disk loop identifier, such as switch setting
d*n*	SCSI lun *n*
s*n*	partition *n* (0-7)

See Also sar(1), format(1M), iostat(1M), ioctl(2), lseek(2), open(2), read(2), write(2), scsi(4)driver.conf(4), cdio(7I), dkio(7I)

ANSI Small Computer System Interface-2 (SCSI-2)

ANSI X3.272-1996, Fibre Channel Arbitrated Loop (FC-AL)

Fibre Channel - Private Loop SCSI Direct Attach (FC-PLDA)

Diagnostics

```
Error for command '<command name>' Error Level: Fatal Requested Block <n>,
Error  Block: <m>, Sense Key: <sense key name>, Vendor '<vendor name>':
ASC = 0x<a> (<ASC name>), ASCQ = 0x<b>, FRU = 0x<c>
```

The command indicated by <command name> failed. The Requested Block is the block where the transfer started and the Error Block is the block that caused the error. Sense Key, ASC, and ASCQ information is returned by the target in response to a request sense command.

```
Check Condition on REQUEST SENSE
```

A REQUEST SENSE command completed with a check condition. The original command will be retried a number of times.

```
Label says <m> blocks Drive says <n> blocks
```

There is a discrepancy between the label and what the drive returned on the READ CAPACITY command.

```
Not enough sense information
```

The request sense data was less than expected.

```
Request Sense couldn't get sense data
```

The REQUEST SENSE command did not transfer any data.

```
Reservation Conflict
```

The drive was reserved by another initiator.

`SCSI transport failed: reason 'xxxx' : {retrying|giving up}`

The host adapter has failed to transport a command to the target for the reason stated. The driver will either retry the command or, ultimately, give up.

`Unhandled Sense Key <`*n*`>`

The REQUEST SENSE data included an invalid sense key.

`Unit not Ready. Additional sense code 0x<`*n*`>`

The drive is not ready.

`corrupt label - bad geometry`

The disk label is corrupted.

`corrupt label - label checksum failed`

The disk label is corrupted.

`corrupt label - wrong magic number`

The disk label is corrupted.

`device busy too long`

The drive returned busy during a number of retries.

`disk not responding to selection`

The drive was probably powered down or died.

`i/o to invalid geometry`

The geometry of the drive could not be established.

`incomplete read/write - retrying/giving up`

There was a residue after the command completed normally.

`logical unit not ready`

The drive is not ready.

`no bp for disk label`

A bp with consistent memory could not be allocated.

`no mem for property`

Free memory pool exhausted.

`no memory for disk label`

Free memory pool exhausted.

`no resources for dumping`

A packet could not be allocated during dumping.

`offline`

Drive went offline; probably powered down.

`requeue of command fails<`*n*`>`

Driver attempted to retry a command and experienced a transport error.

`ssdrestart transport failed <`*n*`>`

Driver attempted to retry a command and experienced a transport error.

`transfer length not modulo <`*n*`>`

Illegal request size.

`transport rejected <`*n*`>`

Host adapter driver was unable to accept a command.

`unable to read label`

Failure to read disk label.

`unit does not respond to selection`

Drive went offline; probably powered down.

Name st – driver for SCSI tape devices

Synopsis `st@`*target*`,`*lun*`:`*l*`,`*m*`,`*h*`,`*c*`,`*ubn*

Description The `st` device driver provides a standard interface to various SCSI tape devices. See `mtio(7I)` for details.

To determine if the `st` device driver supports your tape device, SPARC users should enter the following on a command line:

```
% strings /kernel/drv/sparcv9/st | grep -i <tape device name>
```

x86 users can do the following to determine if the `st` device driver supports a particular tape device:

```
% strings /kernel/drv/st | grep -i <tape device name>
```

The driver can be opened with either rewind on close or no rewind on close options. It can also be opened with the `O_NDELAY` (see `open(2)`) option when there is no tape inserted in the drive. A maximum of four tape formats per device are supported (see `FILES` below). The tape format is specified using the device name. (Tape format is also referred to as tape density).

Following are a list of SCSI commands that can be executed while another host reserves the tape drive. The commands are:

```
SCMD_TEST_UNIT_READY
SCMD_REQUEST_SENSE
SCMD_READ_BLKLIM
SCMD_INQUIRY
SCMD_RESERVE
SCMD_RELEASE
SCMD_DOORLOCK
SCMD_REPORT_DENSITIES
SCMD_LOG_SENSE_G1
SCMD_PERSISTENT_RESERVE_IN
SCMD_PERSISTENT_RESERVE_OUT
SCMD_REPORT_LUNS
```

In multi-initiator environments, the driver will not reserve the tape drive if above commands are issued. For other SCSI commands, the driver reserves the tape drive and releases the drive at close if it has been reserved. Refer to the MTIOCRESERVE and MTIOCRELEASE ioctls in `mtio(7I)` for information about how to allow a tape drive to remain reserved upon close. See the flag options below for information about disabling this feature.

If a SCSI-3 persistent reservation is done through the driver, the driver disables all existing SCSI-2 reservations.

If the tape drive is opened in O_NDELAY mode, no reservation occurs during the open, as per the POSIX standard (see `standards(5)`). However, if a command not found in the above list is used, a reservation will occur to provide reserve/release functionality before the command is issued.

Persistent Errors and Asynchronous Tape Operation

The `st` driver now supports persistent errors (see `mtio(7I)` and asynchronous tape operations (see `mtio(7I)`, `aioread(3C)`, and `aiowrite(3C)`).

Read Operation

If the driver is opened for reading in a different format than the tape is written in, the driver overrides the user-selected format. For example, if a 1/4" cartridge tape is written in QIC-24 format and opened for reading in QIC-150, the driver detects a read failure on the first read and automatically switches to QIC-24 to read the data.

Note that if the low density format is used, no indication is given that the driver has overridden the format you selected. Other formats issue a warning message to inform you of an overridden format selection. Some devices automatically perform this function and do not require driver support (1/2" reel tape drive, for example).

Write Operation

Writing from the beginning of tape is performed in the user-specified format. The original tape format is used for appending onto previously written tapes.

Tape Configuration

The `st` driver has a built-in configuration table for most Sun-supported tape drives. For those tape drives that are not in the table, the `st` driver tries to read the configuration from the tape drive through optional SCSI-3 commands. To support the addition of third party tape devices which are not in the built-in configuration table or not able to report their configuration, device information can be supplied in `st.conf` as global properties that apply to each node, or as properties that are applicable to one node only. By supplying the information in `st.conf`, the built-in configuration is overridden and the `st` driver will not query the configuration from tape drives. The `st` driver looks for the property called `tape-config-list`. The value of this property is a list of triplets, where each triplet consists of three strings.

The formal syntax is:

```
tape-config-list = <triplet> [, <triplet> *];
```

where

```
<triplet> := <vid+pid>, <pretty print>, <data-property-name>
```

and

```
<data-property-name> = <version>, <type>, <bsize>,
        <options>, <number of densities>,
        <density> [, <density>*], <default-density>;
```

or

```
<data-property-name> = <version 2>, <type>, <bsize>,
        <options>, <number of densities>,
        <density> [, <density>*], <default-density>,
        <non-motion time-out>, <I/O time-out>,
        <rewind time-out>, <space time-out>,
        <load time-out>, <unload time-out>,
        <erase time-out>;
```

A semicolon (`;`) is used to terminate a prototype devinfo node specification. Individual elements listed within the specification should not be separated by a semicolon. (Refer to `driver.conf(4)` for more information.)

`<vid+pid>` is the string that is returned by the tape device on a SCSI inquiry command. This string may contain any character in the range `0x20-0x7e`. Characters such as " ' " (double quote) or " ' " (single quote), which are not permitted in property value strings, are represented by their octal equivalent (for example, `\042` and `\047`). Trailing spaces may be truncated.

`<pretty print>` is used to report the device on the console. This string may have zero length, in which case the `<vid+pid>` will be used to report the device.

`<data-property-name>` is the name of the property which contains all the tape configuration values (such as `<type>`, `<bsize>`, etc.) corresponding for the tape drive for the specified `<vid+pid>`.

`<version>` is a version number and should be 1 or 2. In the future, higher version numbers may be used to allow for changes in the syntax of the `<data-property-name>` value list.

`<type>` is a type field. Valid types are defined in `/usr/include/sys/mtio.h`. For third party tape configuration, the following generic types are recommended:

`MT_ISQIC`	0x32
`MT_ISREEL`	0x33
`MT_ISDAT`	0x34
`MT_IS8MM`	0x35
`MT_ISOTHER`	0x36
`MT_ISTAND25G`	0x37
`MT_ISDLT`	0x38
`MT_ISSTK9840`	0x39
`MT_ISBMDLT1`	0x3A

MT_LTO	0x3B

<bsize> is the preferred block size of the tape device. The value should be 0 for variable block size devices.

<options> is a bit pattern representing the devices, as defined in /usr/include/sys/scsi/targets/stdef.h. Valid flags for tape configuration are shown in the following table. Note that this table does not list flags that are non-configurable in st.conf (including ST_KNOWS_MEDIA which uses the media type reported from the mode select data to select the correct density code).

ST_VARIABLE	0x0001
ST_QIC	0x0002
ST_REEL	0x0004
ST_BSF	0x0008
ST_BSR	0x0010
ST_LONG_ERASE	0x0020
ST_AUTODEN_OVERRIDE	0x0040
ST_NOBUF	0x0080
ST_KNOWS_EOD	0x0200
ST_UNLOADABLE	0x0400
ST_SOFT_ERROR_REPORTING	0x0800
ST_LONG_TIMEOUTS	0x1000
ST_NO_RECSIZE_LIMIT	0x8000
ST_MODE_SEL_COMP	0x10000
ST_NO_RESERVE_RELEASE	0x20000
ST_READ_IGNORE_ILI	0x40000
ST_READ_IGNORE_EOFS	0x80000
ST_SHORT_FILEMARKS	0x100000
ST_EJECT_TAPE_ON_CHANGER_FAILURE	0x200000
ST_RETRY_ON_RECOVERED_DEFERRED_ERROR	0x400000
ST_WORMABLE	0x1000000

ST_VARIABLE	The flag indicates the tape device supports variable length record sizes.
ST_QIC	The flag indicates a Quarter Inch Cartridge (QIC) tape device.
ST_REEL	The flag indicates a 1/2–inch reel tape device.
ST_BSF	If flag is set, the device supports backspace over EOF marks (bsf - see mt(1)).
ST_BSR	If flag is set, the tape device supports the backspace record operation (bsr - see mt(1)). If the device does not support bsr, the st driver emulates the action by rewinding the tape and using the forward space record (fsf) operation to forward the tape to the correct file. The driver then uses forward space record (fsr - see mt(1)) to forward the tape to the correct record.
ST_LONG_ERASE	The flag indicates the tape device needs a longer time than normal to erase.
ST_AUTODEN_OVERRIDE	The auto-density override flag. The device is capable of determining the tape density automatically without issuing a "mode-select"/"mode-sense command."
ST_NOBUF	The flag disables the device's ability to perform buffered writes. A buffered write occurs when the device acknowledges the completion of a write request after the data has been written to the device's buffer, but before all of the data has been written to the tape.
ST_KNOWS_EOD	If flag is set, the device can determine when EOD (End of Data) has been reached. When this flag is set, the st driver uses fast file skipping. Otherwise, file skipping happens one file at a time.
ST_UNLOADABLE	The flag indicates the device will not complain if the st driver is unloaded and loaded again (see modload(1M) and modunload(1M)). That is, the driver will return the correct inquiry string.

ST_SOFT_ERROR_REPORTING	The flag indicates the tape device will perform a “request sense” or “log sense” command when the device is closed. Currently, only Exabyte and DAT drives support this feature.
ST_LONG_TIMEOUTS	The flag indicates the tape device requires timeouts that are five times longer than usual for normal operation.
ST_NO_RECSIZE_LIMIT	The flag applies to variable-length tape devices. If this flag is set, the record size is not limited to a 64 Kbyte record size. The record size is only limited by the smaller of either the record size supported by the device or the maximum DMA transfer size of the system. (Refer to Large Record Sizes and WARNINGS.) The maximum block size that will not be broken into smaller blocks can be determined from the mt_bf returned from the MTIOCGET ioctl(). This number is the lesser of the upper block limit returned by the drive from READ BLOCK LIMITS command and the dma-max property set by the Host Bus Adapter (HBA) to which the drive is attached.
ST_MODE_SEL_COMP	If the ST_MODE_SEL_COMP flag is set, the driver determines which of the two mode pages the device supports for selecting or deselecting compression. It first tries the Data Compression mode page (0x0F); if this fails, it tries the Device Configuration mode page (0x10). Some devices, however, may need a specific density code for selecting or deselecting compression. Please refer to the device specific SCSI manual. When the flag is set, compression is enabled only if the "c" or "u" device is used. Note that when the lower 2 densities of a drive are identically configured and the upper 2 densities are identically configured, but the lower and upper differ from each other and ST_MODE_SEL_COMP is set, the “m” node sets compression *on* for the lower density

code (for example, 0x42) and the "c" and "u" nodes set compression *on* for the higher density (for example, 0x43). For any other device densities, compression is disabled.

`ST_NO_RESERVE_RELEASE`

The `ST_NO_RESERVE_RELEASE` flag disables the use of reserve on open and release on close. If an attempt to use a ioctl of `MTRESERVE` or `MTRELEASE` on a drive with this flag set, it will return an error of `ENOTTY` (inappropriate ioctl for device).

`ST_READ_IGNORE_ILI`

The `ST_READ_IGNORE_ILI` flag is applicable only to variable block devices which support the SILI bit option. The `ST_READ_IGNORE_ILI` flag indicates that SILI (supress incorrect length indicator) bit will be set during reads. When this flag is set, short reads (requested read size is less than the record size on the tape) will be successful and the number of bytes transferred will be equal to the record size on the tape. The tape will be positioned at the start of the next record skipping over the extra data (the remaining data has been has been lost). Long reads (requested read size is more than the record size on the tape) will see a large performance gain when this flag is set, due to overhead reduction. When this flag is not set, short reads will return an error of `ENOMEM`.

`ST_READ_IGNORE_EOFS`

The `ST_READ_IGNORE_EOFS` flag is applicable only to 1/2" Reel Tape drives and when performing consecutive reads only. It should not be used for any other tape command. Usually End-of-recorded-media (EOM) is indicated by two EOF marks on 1/2" tape and application cannot read past EOM. When this flag is set, two EOF marks no longer indicate EOM allowing applications to read past two EOF marks. In this case it is the responsibility of the application to detect end-of-recorded-media (EOM). When this flag is set, tape operations (like MTEOM) which positions the tape at

	end-of-recorded-media will fail since detection of end-of-recorded-media (EOM) is to be handled by the application. This flag should be used when backup applications have embedded double filemarks between files.
ST_SHORT_FILEMARKS	The ST_SHORT_FILEMARKS flag is applicable only to EXABYTE 8mm tape drives which supports short filemarks. When this flag is set, short filemarks is used for writing filemarks. Short filemarks could lead to tape incompatible with some otherwise compatible device. By default long filemarks will be used for writing filemarks.
ST_EJECT_TAPE_ON_CHANGER_FAILURE	If ST_EJECT_TAPE_ON_CHANGER_FAILURE flag is set, the tape is ejected automatically if the tape cartridge is trapped in the medium due to positioning problems of the medium changer. The following ASC/ASCQ keys are defined to the reasons for causing tape ejection if ST_EJECT_TAPE_ON_CHANGER_FAILURE option is set to 0x200000: Sense ASC/ASCQ Description Key 4 15/01 Mechanical Failure 4 44/00 Internal Target Failure 2 53/00 Media Load or Eject Failed 4 53/00 Media Load or Eject Failed 4 53/01 Unload Tape Failure
ST_RETRY_ON_RECOVERED_DEFERRED_ERROR	If ST_RETRY_ON_RECOVERED_DEFERRED_ERROR flag is set, the st driver will retry the last write if this cmd caused a check condition with

	error code 0x71 and sense code 0x01. Some tape drives, notably the IBM 3090, require this option.
`ST_WORMABLE`	When `ST_WORMABLE` is set, `st` attempts to detect the presence of WORM media in the device.

<number of densities> is the number of densities specified. Each tape drive can support up to four densities. The value entered should therefore be between 1 and 4; if less than 4, the remaining densities will be assigned a value of 0x0.

<density> is a single-byte hexadecimal number. It can either be found in the device specification manual or be obtained from the device vendor.

<default-density> has a value between 0 and (<number of densities> - 1).

<non-motion time-out> Time in seconds that the drive should be able to perform any SCSI command that doesn't require tape to be moved. This includes mode sense, mode select, reserve, release, read block limits, and test unit ready.

<I/O time-out> Time in seconds to perform data transfer I/O to or from tape including worst case error recovery.

<rewind time-out> Time in seconds to rewind from anywhere on tape to BOT including worst case recovery forcing buffered write data to tape.

<space time-out> Time in seconds to space to any file, block or end of data on tape. Including worst case when any form of cataloging is invalid.

<load time-out> Time in seconds to load tape and be ready to transfer first block. This should include worst case recovery reading tape catalog or drive specific operations done at load.

<unload time-out> Time in seconds to unload tape. Should include worst case time to write to catalog, unthread, and tape cartridge unloading. Also should include worst case time for any drive specific operations that are preformed at unload. Should not include rewind time as the driver rewinds tape before issuing the unload.

<erase time-out> Time in seconds to preform a full (BOT to EOT) erase of longest medium with worst case error recovery.

Device Statistics Support

Each device maintains I/O statistics both for the device and for each partition allocated on that device. For each device/partition, the driver accumulates reads, writes, bytes read, and bytes written. The driver also takes hi-resolution time stamps at queue entry and exit points, which facilitates monitoring the residence time and cumulative residence-length product for each queue.

Each device also has error statistics associated with it. These must include counters for hard errors, soft errors and transport errors. Other data may be implemented as required.

ioctls The behavior of SCSI tape positioning ioctls is the same across all devices which support them. (Refer to `mtio(7I)`.) However, not all devices support all ioctls. The driver returns an `ENOTTY` error on unsupported ioctls.

The retension ioctl only applies to 1/4" cartridge tape devices. It is used to restore tape tension, thus improving the tape's soft error rate after extensive start-stop operations or long-term storage.

In order to increase performance of variable-length tape devices (particularly when they are used to read/write small record sizes), two operations in the `MTIOCTOP` ioctl, `MTSRSZ` and `MTGRSZ`, can be used to set and get fixed record lengths. The ioctl also works with fixed-length tape drives which allow multiple record sizes. The min/max limits of record size allowed on a driver are found by using a SCSI-2 `READ BLOCK LIMITS` command to the device. If this command fails, the default min/max record sizes allowed are 1 byte and 63k bytes. An application that needs to use a different record size opens the device, sets the size with the `MTSRSZ` ioctl, and then continues with I/O. The scope of the change in record size remains until the device is closed. The next open to the device resets the record size to the default record size (retrieved from `st.conf`).

Note that the error status is reset by the `MTIOCGET` get status ioctl call or by the next read, write, or other ioctl operation. If no error has occurred (sense key is `0`), the current file and record position is returned.

Errors

`EACCES` The driver is opened for write access and the tape is write-protected or the tape unit is reserved by another host.

`EBUSY` The tape drive is in use by another process. Only one process can use the tape drive at a time. The driver will allow a grace period for the other process to finish before reporting this error.

`EINVAL` The number of bytes read or written is not a multiple of the physical record size (fixed-length tape devices only).

`EIO` During opening, the tape device is not ready because either no tape is in the drive, or the drive is not on-line. Once open, this error is returned if the requested I/O transfer could not be completed.

`ENOTTY` This indicates that the tape device does not support the requested ioctl function.

`ENXIO` During opening, the tape device does not exist.

`ENOMEM` This indicates that the record size on the tape drive is more than the requested size during read operation.

Examples

EXAMPLE 1 Global tape-config list property

The following is an example of a global `tape-config-list` property:

```
tape-config-list =
"Magic DAT", "Magic 4mm Helical Scan", "magic-data",
"Major Appliance", "Major Appliance Tape", "major-tape";


magic-data  = 1,0x34,1024,0x1639,4,0,0x8c,0x8c,0x8c,3;
major-tape = 2,0x3c,0,0x18619,4,0x0,0x0,0x0,0x0,
             3,0,0,30,120,0,0,36000;

name="st" class="scsi"
          target=0 lun=0;
name="st" class="scsi"
          target=1 lun=0;
name="st" class="scsi"
          target=2 lun=0;
          .
          .
          .
name="st" class="scsi"
          target=6 lun=0;
```

EXAMPLE 2 Tape-config-list property applicable to target 2 only

The following is an example of a tape-config-list property applicable to target 2 only:

```
name="st" class="scsi"
        target=0 lun=0;
name="st" class="scsi"
        target=1 lun=0;
name="st" class="scsi"
        target=2 lun=0
        tape-config-list =
        "Magic   DAT", "Magic 4mm Helical Scan", "magic-data"
        magic-data = 1,0x34,1024,0x1639,4,0,0x8c,0x8c,0x8c,3;
name="st" class="scsi"
        target=3 lun=0;
          .
          .
          .
name="st" class="scsi"
        target=6 lun=0;
```

Large Record Sizes

To support applications such as seismic programs that require large record sizes, the flag `ST_NO_RECSIZE_LIMIT` must be set in drive option in the configuration entry. A SCSI tape drive that needs to transfer large records should OR this flag with other flags in the 'options' field in `st.conf`. (Refer to `Tape Configuration`.) By default, this flag is set for the built-in config entries of Archive DAT and Exabyte drives.

If this flag is set, the `st` driver issues a SCSI-2 `READ BLOCK LIMITS` command to the device to determine the maximum record size allowed by it. If the command fails, `st` continues to use the maximum record sizes mentioned in the `mtio(7I)` man page.

If the command succeeds, `st` restricts the maximum transfer size of a variable-length device to the minimum of that record size and the maximum DMA size that the host adapter can handle. Fixed-length devices are bound by the maximum DMA size allocated by the machine. Note that tapes created with a large record size may not be readable by earlier releases or on other platforms.

(Refer to the `WARNINGS` section for more information.)

EOT Handling

The Emulex drives have only a physical end of tape (PEOT); thus it is not possible to write past EOT. All other drives have a logical end of tape (LEOT) before PEOT to guarantee flushing the data onto the tape. The amount of storage between LEOT and PEOT varies from less than 1 Mbyte to about 20 Mbyte, depending on the tape drive.

If EOT is encountered while writing an Emulex, no error is reported but the number of bytes transferred is `0` and no further writing is allowed. On all other drives, the first write that encounters EOT will return a short count or `0`. If a short count is returned, then the next write will return `0`. After a zero count is returned, the next write returns a full count or short count. A following write returns `0` again. It is important that the number and size of trailer records be kept as small as possible to prevent data loss. Therefore, writing after EOT is not recommended.

Reading past EOT is transparent to the user. Reading is stopped only by reading EOF's. For 1/2” reel devices, it is possible to read off the end of the reel if one reads past the two file marks which mark the end of recorded media.

Files

`/kernel/drv/st.conf`	driver configuration file
`/usr/include/sys/mtio.h`	structures and definitions for mag tape io control commands
`/usr/include/sys/scsi/targets/stdef.h`	definitions for SCSI tape drives
`/dev/rmt/[0– 127][l,m,h,u,c][b][n]`	where `l,m,h,u,c` specifies the density (low, medium, high, ultra/compressed), `b` the optional BSD behavior (see `mtio(7I)`), and `n` the optional no rewind behavior. For

example, `/dev/rmt/0lbn` specifies unit 0, low density, BSD behavior, and no rewind.

For 1/2" reel tape devices (HP-88780), the densities are:

`l`	800 BPI density
`m`	1600 BPI density
`h`	6250 BPI density
`c`	data compression (not supported on all modules)

For 8mm tape devices (Exabyte 8200/8500/8505):

`l`	Standard 2 Gbyte format
`m`	5 Gbyte format (8500, 8505 only)
`h,c`	5 Gbyte compressed format (8505 only)

For 4mm DAT tape devices (Archive Python):

`l`	Standard format
`m,h,c`	data compression

For all QIC (other than QIC-24) tape devices:

`l,m,h,c`	density of the tape cartridge type (not all devices can read and

	write all formats)

For QIC-24 tape devices (Emulex MT–02):

`l`	QIC-11 Format
`m,h,c`	QIC-24 Format

See Also `mt(1)`, `modload(1M)`, `modunload(1M)`, `open(2)`, `read(2)`, `write(2)`, `aioread(3C)`, `aiowrite(3C)`, `kstat(3KSTAT)`, `driver.conf(4)`, `scsi(4)`, `standards(5)`, `esp(7D)`, `isp(7D)`, `mtio(7I)`, `ioctl(9E)`

Diagnostics The `st` driver diagnostics may be printed to the console or messages file.

Each diagnostic is dependent on the value of the system variable `st_error_level`. `st_error_level` may be set in the `/etc/system` file. The default setting for `st_error_level` is 4 (SCSI_ERR_RETRYABLE) which is suitable for most configurations since only actual fault diagnostics are printed. Settings range from values 0 (SCSI_ERR_ALL) which is most verbose, to 6 (SCSI_ERR_NONE) which is least verbose. See `stdef.h` for the full list of error-levels. SCSI_ERR_ALL level the amount of diagnostic information is likely to be excessive and unnecessary.

The st driver diagnostics are described below:

```
Error for Command: <scsi_cmd_name()> Error Level:<error_class>
Requested Block: <blkno>  Error Block: <err_blkno>
Vendor: <name>: Serial Number: <inq_serial>
Sense Key: <es_key> ASC: 0x<es_add_code> (scsi_asc_ascq_name()>), ASCQ:
0x<es_qual_code>, FRU: 0x<ex_fru_code>
```

where <error_class> may be any one of the following: "All," "Unknown," "Informational," "Recovered," "Retryable," "Fatal"

The command indicated by <scsi_cmd_name> failed. Requested Block represents the block where the transfer started. Error Block represents the block that caused the error. Sense Key, ASC, ASCQ and FRU information is returned by the target in response to a request sense command. See SCSI protocol documentation for description of Sense Key, ASC, ASCQ, FRU.

The st driver attempts to validate entries in the `st.conf` file. Each field in the entry is checked for upper and lower limits and invalid bits set. The fields are named as follows in config string order:

```
conf version
drive type
block size
options
number of densities
```

```
density code
default density
non motion timeout
I/O timeout
space timeout
load timeout
unload timeout
erase timeout
```

The `st.conf` diagnostics are described below:

```
<con-name> <field-in-err> <problem-with-field>
```

where <con-name> is the name of the config string. Where <field-in-err> is the field containing invalid entries and where <problem-with-field> describes the nature of the invalid entry.

```
Write/read: not modulo <n> block size
```

The request size for fixed record size devices must be a multiple of the specified block size.

```
Recovery by resets failed
```

After a transport error, the driver attempted to recover by issuing a device reset and then a bus reset if device reset failed. These recoveries failed.

```
Periodic head cleaning required
```

The driver reported that periodic head cleaning is now required. This diagnostic is generated either due to a threshold number of retries, or due to the device communicating to the driver that head cleaning is required.

```
Soft error rate (<n>%) during writing/reading was too high
```

The soft error rate has exceeded the threshold specified by the vendor.

```
SCSI transport failed: reason 'xxxx': {retrying|giving up}
```

The Host Bus Adapter (HBA) has failed to transport a command to the target for the reason stated. The driver will either retry the command or, ultimately, give up.

```
Tape not inserted in drive
```

A media access command was attempted while there was no tape inserted into the specified drive. In this case, the drive returns sense key of DRIVE NOT READY.

```
Transport rejected
```

The Host Bus Adapter (HBA) driver is not accepting commands after failing to successfully transport a scsi packet to the target. The actual status received by the `st` driver from the underlying HBA driver was either TRAN_FATAL_ERROR or TRAN_BADPKT.

```
Retrying command
```

The st driver failed to complete a command. However the command is retryable and will be retried.

```
Giving up
```

The st driver has exhausted retries or otherwise is unable to retry the command and so is giving up.

```
No target struct for st%d
```

The st driver failed to obtain state information because the requested state structure was not allocated. The specified device was probably not attached.

```
File mark detected
```

The operation detected an end of file mark. (File marks signify the end of a file on the tape media).

```
End-of-media detected
```

The operation reached the end of the tape media.

```
Exabyte soft error reporting failed. DAT soft error reporting failed
```

The st driver was unable to determine if the soft error threshold had been exceeded because it did not successfully read the data it requires or did not obtain enough data. This data is retrieved using the log sense command.

```
Log sense parameter code does not make sense
```

The log sense command retrieves hardware statistics that are stored on the drive (for example, soft error counts and retries.) If the data retrieved from the drive is invalid, this message is printed and the data is not used.

```
Restoring tape position at fileno=%x, blkno=%lx....
```

The st driver is positioning to the specified file and block. This occurs on an open.

```
Failed to restore the last <file/block> position:
In this state, tape will be loaded at BOT during next open
```

The st driver could not position to the specified location and will revert to the beginning of the tape when the next open is attempted.

```
Device does not support compression
```

The compression facility of the device was requested. However the device does not have a hardware compression capability.

```
DAT soft error reset failed
```

After DAT soft error reporting, the counters within the device that accumulate this sense data need to be re-set. This operation failed.

```
Errors after pkt alloc (b_flags=0x%x, b_error=0x%x)
```

Memory allocation for a scsi packet failed.

```
Incorrect length indicator set
```

The drive reported the length of data requested in a READ operation, was incorrect. Incorrect Length Indicator (ILI) is a very commonly used facility in SCSI tape protocol and should not be seen as an error per-se. Applications typically probe a new tape with a read of any length, using the returned length to the read system call for future reads. Along with this operation, an underlying ILI error is received. ILI errors are therefore informational only and are masked at the default `st_error_level`.

```
Data property (%s) has no value
Data property (%s) incomplete
Version # for data property (%s) greater than 1
```

These diagnostics indicate problems in retrieving the values of the various property settings. The `st` driver is in the process of setting the property/parameter values for the tape drive using information from either the built-in table within the driver or from uncommented entries in the `st.conf` file. The effect on the system may be that the tape drive may be set with default or generic driver settings which may not be appropriate for the actual type of tape drive being used.

```
st_attach-RESUME: tape failure tape position will be lost
```

On a resume after a power management suspend, the previously known tape position is no longer valid. This can occur if the tape was changed while the system was in power management suspend. The operation will not be retried.

```
Write Data Buffering has been deprecated. Your applications should
continue to work normally. However, they should be ported to use
Asynchronous I/O.
```

Indicates that buffering has been removed from Solaris.

```
Cannot detach: fileno=%x, blkno=%lx
```

The `st` driver cannot unload because the tape is not positioned at BOT (beginning of tape). May indicate hardware problems with the tape drive.

```
Variable record length I/O
Fixed record length (%d byte blocks) I/O
```

Tape-drives can use either Fixed or Variable record length. If the drive uses Fixed length records, then the built in property table or the st.conf file will contain a non-zero record-length property. Most DAT, Exabyte and DLT drives support Variable record lengths. Many QIC format tape drives have historically been of Fixed record length.

```
Command will be retried
 un_ncmds: %d can't retry cmd
```

These diagnostics are only seen with tape drives with the ST_RETRY_ON_RECOVERED_DEFERRED_ERROR bit set. See `stdef.h` for explanation of the specific usage of this setting.

Warnings Effective with Solaris 2.4, the `ST_NO_RECSIZE_LIMIT` flag is set for the built-in config entries of the Archive DAT and Exabyte drivers by default. (Refer to `Large Record Sizes`.) Tapes written with large block sizes prior to Solaris 2.4 may cause some applications to fail if the number of bytes returned by a read request is less than the requested block size (for example, asking for 128 Kbytes and receiving less than 64 Kbytes).

The `ST_NO_RECSIZE_LIMIT` flag can be disabled in the config entry for the device as a work-around. (Refer to `Tape Configuration`.) This action disables the ability to read and write with large block sizes and allows the reading of tapes written prior to Solaris 2.4 with large block sizes.

(Refer to `mtio(7I)` for a description of maximum record sizes.)

Bugs Tape devices that do not return a BUSY status during tape loading prevent user commands from being held until the device is ready. The user must delay issuing any tape operations until the tape device is ready. This is not a problem for tape devices supplied by Sun Microsystems.

Tape devices that do not report a blank check error at the end of recorded media may cause file positioning operations to fail. Some tape drives, for example, mistakenly report media error instead of blank check error.

Name streamio – STREAMS ioctl commands

Synopsis

```
#include <sys/types.h>
#include <stropts.h>
#include <sys/conf.h>

int ioctl(int fildes, int command, ... /*arg*/);
```

Description STREAMS (see Intro(3)) `ioctl` commands are a subset of the `ioctl(2)` commands and perform a variety of control functions on streams.

The *fildes* argument is an open file descriptor that refers to a stream. The *command* argument determines the control function to be performed as described below. The *arg* argument represents additional information that is needed by this command. The type of *arg* depends upon the command, but it is generally an integer or a pointer to a command-specific data structure. The *command* and *arg* arguments are interpreted by the STREAM head. Certain combinations of these arguments may be passed to a module or driver in the stream.

Since these STREAMS commands are `ioctl`s, they are subject to the errors described in `ioctl(2)`. In addition to those errors, the call will fail with `errno` set to `EINVAL`, without processing a control function, if the STREAM referenced by *fildes* is linked below a multiplexor, or if *command* is not a valid value for a stream.

Also, as described in `ioctl(2)`, STREAMS modules and drivers can detect errors. In this case, the module or driver sends an error message to the STREAM head containing an error value. This causes subsequent calls to fail with `errno` set to this value.

ioctls The following `ioctl` commands, with error values indicated, are applicable to all STREAMS files:

`I_PUSH` Pushes the module whose name is pointed to by *arg* onto the top of the current stream, just below the STREAM head. If the STREAM is a pipe, the module will be inserted between the stream heads of both ends of the pipe. It then calls the open routine of the newly-pushed module. On failure, `errno` is set to one of the following values:

- `EINVAL` Invalid module name.
- `EFAULT` *arg* points outside the allocated address space.
- `ENXIO` Open routine of new module failed.
- `ENXIO` Hangup received on *fildes*.
- `ENOTSUP` Pushing a module is not supported on this stream.

`I_POP` Removes the module just below the STREAM head of the STREAM pointed to by *fildes*. To remove a module from a pipe requires that the module was pushed on the side it is being removed from. *arg* should be `0` in an `I_POP` request. On failure, `errno` is set to one of the following values:

EINVAL No module present in the stream.

ENXIO Hangup received on *fildes*.

EPERM Attempt to pop through an anchor by an unprivileged process.

ENOTSUP Removal is not supported.

I_ANCHOR Positions the stream anchor to be at the stream's module directly below the stream head. Once this has been done, only a privileged process may pop modules below the anchor on the stream. *arg* must be `0` in an `I_ANCHOR` request. On failure, `errno` is set to the following value:

EINVAL Request to put an anchor on a pipe.

I_LOOK Retrieves the name of the module just below the stream head of the stream pointed to by *fildes*, and places it in a null terminated character string pointed at by *arg*. The buffer pointed to by *arg* should be at least `FMNAMESZ`+1 bytes long. This requires the declaration `#include <sys/conf.h>`. On failure, `errno` is set to one of the following values:

EFAULT *arg* points outside the allocated address space.

EINVAL No module present in stream.

I_FLUSH This request flushes all input and/or output queues, depending on the value of *arg*. Legal *arg* values are:

FLUSHR Flush read queues.

FLUSHW Flush write queues.

FLUSHRW Flush read and write queues.

If a pipe or FIFO does not have any modules pushed, the read queue of the stream head on either end is flushed depending on the value of *arg*.

If `FLUSHR` is set and *fildes* is a pipe, the read queue for that end of the pipe is flushed and the write queue for the other end is flushed. If *fildes* is a FIFO, both queues are flushed.

If `FLUSHW` is set and *fildes* is a pipe and the other end of the pipe exists, the read queue for the other end of the pipe is flushed and the write queue for this end is flushed. If *fildes* is a FIFO, both queues of the FIFO are flushed.

If `FLUSHRW` is set, all read queues are flushed, that is, the read queue for the FIFO and the read queue on both ends of the pipe are flushed.

Correct flush handling of a pipe or FIFO with modules pushed is achieved via the `pipemod` module. This module should be the first module pushed onto a pipe so that it is at the midpoint of the pipe itself.

On failure, `errno` is set to one of the following values:

`ENOSR` Unable to allocate buffers for flush message due to insufficient stream memory resources.

`EINVAL` Invalid *arg* value.

`ENXIO` Hangup received on *fildes*.

`I_FLUSHBAND` Flushes a particular band of messages. *arg* points to a `bandinfo` structure that has the following members:

```
unsigned char bi_pri;
int bi_flag;
```

The `bi_flag` field may be one of `FLUSHR`, `FLUSHW`, or `FLUSHRW` as described earlier.

`I_SETSIG` Informs the stream head that the user wishes the kernel to issue the `SIGPOLL` signal (see `signal(3C)`) when a particular event has occurred on the stream associated with *fildes*. `I_SETSIG` supports an asynchronous processing capability in streams. The value of *arg* is a bitmask that specifies the events for which the user should be signaled. It is the bitwise OR of any combination of the following constants:

`S_INPUT` Any message other than an `M_PCPROTO` has arrived on a stream head read queue. This event is maintained for compatibility with previous releases. This event is triggered even if the message is of zero length.

`S_RDNORM` An ordinary (non-priority) message has arrived on a stream head read queue. This event is triggered even if the message is of zero length.

`S_RDBAND` A priority band message (band > 0) has arrived on a stream head read queue. This event is triggered even if the message is of zero length.

`S_HIPRI` A high priority message is present on the stream head read queue. This event is triggered even if the message is of zero length.

`S_OUTPUT` The write queue just below the stream head is no longer full. This notifies the user that there is room on the queue for sending (or writing) data downstream.

`S_WRNORM` This event is the same as `S_OUTPUT`.

`S_WRBAND` A priority band greater than 0 of a queue downstream exists and is writable. This notifies the user that there is room on the queue for sending (or writing) priority data downstream.

S_MSG — A STREAMS signal message that contains the SIGPOLL signal has reached the front of the stream head read queue.

S_ERROR — An M_ERROR message has reached the stream head.

S_HANGUP — An M_HANGUP message has reached the stream head.

S_BANDURG — When used in conjunction with S_RDBAND, SIGURG is generated instead of SIGPOLL when a priority message reaches the front of the stream head read queue.

A user process may choose to be signaled only of high priority messages by setting the *arg* bitmask to the value S_HIPRI.

Processes that wish to receive SIGPOLL signals must explicitly register to receive them using I_SETSIG. If several processes register to receive this signal for the same event on the same stream, each process will be signaled when the event occurs.

If the value of *arg* is zero, the calling process will be unregistered and will not receive further SIGPOLL signals. On failure, errno is set to one of the following values:

EINVAL — *arg* value is invalid or *arg* is zero and process is not registered to receive the SIGPOLL signal.

EAGAIN — Allocation of a data structure to store the signal request failed.

I_GETSIG — Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal. The events are returned as a bitmask pointed to by *arg*, where the events are those specified in the description of I_SETSIG above. On failure, errno is set to one of the following values:

EINVAL — Process not registered to receive the SIGPOLL signal.

EFAULT — *arg* points outside the allocated address space.

I_FIND — Compares the names of all modules currently present in the stream to the name pointed to by *arg*, and returns 1 if the named module is present in the stream. It returns 0 if the named module is not present. On failure, errno is set to one of the following values:

EFAULT — *arg* points outside the allocated address space.

EINVAL — *arg* does not contain a valid module name.

I_PEEK — Allows a user to retrieve the information in the first message on the stream head read queue without taking the message off the queue. I_PEEK is

analogous to getmsg(2) except that it does not remove the message from the queue. *arg* points to a strpeek structure, which contains the following members:

```
struct strbuf ctlbuf;
struct strbuf    databuf;
long flags;
```

The maxlen field in the ctlbuf and databuf strbuf structures (see getmsg(2)) must be set to the number of bytes of control information and/or data information, respectively, to retrieve. flags may be set to RS_HIPRI or 0. If RS_HIPRI is set, I_PEEK will look for a high priority message on the stream head read queue. Otherwise, I_PEEK will look for the first message on the stream head read queue.

I_PEEK returns 1 if a message was retrieved, and returns 0 if no message was found on the stream head read queue. It does not wait for a message to arrive. On return, ctlbuf specifies information in the control buffer, databuf specifies information in the data buffer, and flags contains the value RS_HIPRI or 0. On failure, errno is set to the following value:

EFAULT — *arg* points, or the buffer area specified in ctlbuf or databuf is, outside the allocated address space.

EBADMSG — Queued message to be read is not valid for I_PEEK.

EINVAL — Illegal value for flags.

ENOSR — Unable to allocate buffers to perform the I_PEEK due to insufficient STREAMS memory resources.

I_SRDOPT — Sets the read mode (see read(2)) using the value of the argument *arg*. Legal *arg* values are:

RNORM — Byte-stream mode, the default.

RMSGD — Message-discard mode.

RMSGN — Message-nondiscard mode.

In addition, the stream head's treatment of control messages may be changed by setting the following flags in *arg*:

RPROTNORM — Reject read() with EBADMSG if a control message is at the front of the stream head read queue.

RPROTDAT — Deliver the control portion of a message as data when a user issues read(). This is the default behavior.

RPROTDIS — Discard the control portion of a message, delivering any data portion, when a user issues a read( ).

On failure, `errno` is set to the following value:

`EINVAL` *arg* is not one of the above legal values, or *arg* is the bitwise inclusive OR of `RMSGD` and `RMSGN`.

`I_GRDOPT` Returns the current read mode setting in an `int` pointed to by the argument *arg*. Read modes are described in `read( )`. On failure, `errno` is set to the following value:

`EFAULT` *arg* points outside the allocated address space.

`I_NREAD` Counts the number of data bytes in data blocks in the first message on the stream head read queue, and places this value in the location pointed to by *arg*. The return value for the command is the number of messages on the stream head read queue. For example, if zero is returned in *arg*, but the `ioctl` return value is greater than zero, this indicates that a zero-length message is next on the queue. On failure, `errno` is set to the following value:

`EFAULT` *arg* points outside the allocated address space.

`I_FDINSERT` Creates a message from specified buffer(s), adds information about another stream and sends the message downstream. The message contains a control part and an optional data part. The data and control parts to be sent are distinguished by placement in separate buffers, as described below.

The *arg* argument points to a `strfdinsert` structure, which contains the following members:

```
struct  strbuf  ctlbuf;
struct  strbuf    databuf;
t_uscalar_t  flags;
int  fildes;
int  offset;
```

The `len` member in the `ctlbuf` `strbuf` structure (see `putmsg(2)`) must be set to the size of a `t_uscalar_t` plus the number of bytes of control information to be sent with the message. The `fildes` member specifies the file descriptor of the other stream, and the `offset` member, which must be suitably aligned for use as a `t_uscalar_t`, specifies the offset from the start of the control buffer where `I_FDINSERT` will store a `t_uscalar_t` whose interpretation is specific to the stream end. The `len` member in the `databuf` `strbuf` structure must be set to the number of bytes of data information to be sent with the message, or to 0 if no data part is to be sent.

The `flags` member specifies the type of message to be created. A normal message is created if `flags` is set to 0, and a high-priority message is created if `flags` is set to `RS_HIPRI`. For non-priority messages, `I_FDINSERT` will block if the stream write queue is full due to internal flow control

conditions. For priority messages, I_FDINSERT does not block on this condition. For non-priority messages, I_FDINSERT does not block when the write queue is full and O_NDELAY or O_NONBLOCK is set. Instead, it fails and sets errno to EAGAIN.

I_FDINSERT also blocks, unless prevented by lack of internal resources, waiting for the availability of message blocks in the stream, regardless of priority or whether O_NDELAY or O_NONBLOCK has been specified. No partial message is sent.

The ioctl() function with the I_FDINSERT command will fail if:

EAGAIN A non-priority message is specified, the O_NDELAY or O_NONBLOCK flag is set, and the stream write queue is full due to internal flow control conditions.

ENOSR Buffers can not be allocated for the message that is to be created.

EFAULT The *arg* argument points, or the buffer area specified in ctlbuf or databuf is, outside the allocated address space.

EINVAL One of the following: The fildes member of the strfdinsert structure is not a valid, open stream file descriptor; the size of a t_uscalar_t plus offset is greater than the len member for the buffer specified through ctlptr; the offset member does not specify a properly-aligned location in the data buffer; or an undefined value is stored in flags.

ENXIO Hangup received on the fildes argument of the ioctl call or the fildes member of the strfdinsert structure.

ERANGE The len field for the buffer specified through databuf does not fall within the range specified by the maximum and minimum packet sizes of the topmost stream module; or the len member for the buffer specified through databuf is larger than the maximum configured size of the data part of a message; or the len member for the buffer specified through ctlbuf is larger than the maximum configured size of the control part of a message.

I_FDINSERT can also fail if an error message was received by the stream head of the stream corresponding to the fildes member of the strfdinsert structure. In this case, errno will be set to the value in the message.

I_STR Constructs an internal STREAMS ioctl message from the data pointed to by *arg*, and sends that message downstream.

This mechanism is provided to send user `ioctl` requests to downstream modules and drivers. It allows information to be sent with the `ioctl`, and will return to the user any information sent upstream by the downstream recipient. `I_STR` blocks until the system responds with either a positive or negative acknowledgement message, or until the request times out after some period of time. If the request times out, it fails with `errno` set to `ETIME`.

To send requests downstream, *arg* must point to a `strioctl` structure which contains the following members:

```
int  ic_cmd;
int  ic_timout;
int  ic_len;
char  *ic_dp;
```

`ic_cmd` is the internal `ioctl` command intended for a downstream module or driver and `ic_timout` is the number of seconds (-1 = infinite, 0 = use default, >0 = as specified) an `I_STR` request will wait for acknowledgement before timing out. `ic_len` is the number of bytes in the data argument and `ic_dp` is a pointer to the data argument. The `ic_len` field has two uses: on input, it contains the length of the data argument passed in, and on return from the command, it contains the number of bytes being returned to the user (the buffer pointed to by `ic_dp` should be large enough to contain the maximum amount of data that any module or the driver in the stream can return).

At most one `I_STR` can be active on a stream. Further `I_STR` calls will block until the active `I_STR` completes via a positive or negative acknowlegment, a timeout, or an error condition at the stream head. By setting the `ic_timout` field to 0, the user is requesting STREAMS to provide the `DEFAULT` timeout. The default timeout is specific to the STREAMS implementation and may vary depending on which release of Solaris you are using. For Solaris 8 (and earlier versions), the default timeout is fifteen seconds. The `O_NDELAY` and `O_NONBLOCK` (see open(2)) flags have no effect on this call.

The stream head will convert the information pointed to by the `strioctl` structure to an internal `ioctl` command message and send it downstream. On failure, `errno` is set to one of the following values:

`ENOSR` Unable to allocate buffers for the `ioctl` message due to insufficient STREAMS memory resources.

`EFAULT` Either *arg* points outside the allocated address space, or the buffer area specified by `ic_dp` and `ic_len` (separately for data sent and data returned) is outside the allocated address space.

EINVAL — `ic_len` is less than 0 or `ic_len` is larger than the maximum configured size of the data part of a message or `ic_timout` is less than -1.

ENXIO — Hangup received on *fildes*.

ETIME — A downstream `ioctl` timed out before acknowledgement was received.

An `I_STR` can also fail while waiting for an acknowledgement if a message indicating an error or a hangup is received at the stream head. In addition, an error code can be returned in the positive or negative acknowledgement message, in the event the ioctl command sent downstream fails. For these cases, `I_STR` will fail with `errno` set to the value in the message.

`I_SWROPT` — Sets the write mode using the value of the argument *arg*. Legal bit settings for *arg* are:

SNDZERO — Send a zero-length message downstream when a write of 0 bytes occurs.

To not send a zero-length message when a write of 0 bytes occurs, this bit must not be set in *arg*.

On failure, `errno` may be set to the following value:

EINVAL — *arg* is not the above legal value.

`I_GWROPT` — Returns the current write mode setting, as described above, in the `int` that is pointed to by the argument *arg*.

`I_SENDFD` — Requests the stream associated with *fildes* to send a message, containing a file pointer, to the stream head at the other end of a stream pipe. The file pointer corresponds to *arg*, which must be an open file descriptor.

`I_SENDFD` converts *arg* into the corresponding system file pointer. It allocates a message block and inserts the file pointer in the block. The user id and group id associated with the sending process are also inserted. This message is placed directly on the read queue (see `Intro(3)`) of the stream head at the other end of the stream pipe to which it is connected. On failure, `errno` is set to one of the following values:

EAGAIN — The sending stream is unable to allocate a message block to contain the file pointer.

EAGAIN — The read queue of the receiving stream head is full and cannot accept the message sent by `I_SENDFD`.

EBADF — *arg* is not a valid, open file descriptor.

EINVAL *fildes* is not connected to a stream pipe.

ENXIO Hangup received on *fildes*.

I_RECVFD Retrieves the file descriptor associated with the message sent by an I_SENDFD ioctl over a stream pipe. *arg* is a pointer to a data buffer large enough to hold an strrecvfd data structure containing the following members:

```
int  fd;
uid_t  uid;
gid_t  gid;
```

fd is an integer file descriptor. uid and gid are the user id and group id, respectively, of the sending stream.

If O_NDELAY and O_NONBLOCK are clear (see open(2)), I_RECVFD will block until a message is present at the stream head. If O_NDELAY or O_NONBLOCK is set, I_RECVFD will fail with errno set to EAGAIN if no message is present at the stream head.

If the message at the stream head is a message sent by an I_SENDFD, a new user file descriptor is allocated for the file pointer contained in the message. The new file descriptor is placed in the fd field of the strrecvfd structure. The structure is copied into the user data buffer pointed to by *arg*. On failure, errno is set to one of the following values:

EAGAIN A message is not present at the stream head read queue, and the O_NDELAY or O_NONBLOCK flag is set.

EBADMSG The message at the stream head read queue is not a message containing a passed file descriptor.

EFAULT *arg* points outside the allocated address space.

EMFILE NOFILES file descriptors are currently open.

ENXIO Hangup received on *fildes*.

EOVERFLOW *uid* or *gid* is too large to be stored in the structure pointed to by *arg*.

I_LIST Allows the user to list all the module names on the stream, up to and including the topmost driver name. If *arg* is NULL, the return value is the number of modules, including the driver, that are on the stream pointed to by *fildes*. This allows the user to allocate enough space for the module names. If *arg* is non-null, it should point to an str_list structure that has the following members:

```
int sl_nmods;
struct  str_mlist  *sl_modlist;
```

The `str_mlist` structure has the following member:

```
char l_name[FMNAMESZ+1];
```

The `sl_nmods` member indicates the number of entries the process has allocated in the array. Upon return, the `sl_modlist` member of the `str_list` structure contains the list of module names, and the number of entries that have been filled into the `sl_modlist` array is found in the `sl_nmods` member (the number includes the number of modules including the driver). The return value from `ioctl()` is 0. The entries are filled in starting at the top of the stream and continuing downstream until either the end of the stream is reached, or the number of requested modules (`sl_nmods`) is satisfied. On failure, `errno` may be set to one of the following values:

`EINVAL` The `sl_nmods` member is less than 1.

`EAGAIN` Unable to allocate buffers

`I_ATMARK` Allows the user to see if the current message on the stream head read queue is "marked" by some module downstream. *arg* determines how the checking is done when there may be multiple marked messages on the stream head read queue. It may take the following values:

`ANYMARK` Check if the message is marked.

`LASTMARK` Check if the message is the last one marked on the queue.

The return value is `1` if the mark condition is satisfied and `0` otherwise. On failure, `errno` is set to the following value:

`EINVAL` Invalid *arg* value.

`I_CKBAND` Check if the message of a given priority band exists on the stream head read queue. This returns `1` if a message of a given priority exists, `0` if not, or `−1` on error. *arg* should be an integer containing the value of the priority band in question. On failure, `errno` is set to the following value:

`EINVAL` Invalid *arg* value.

`I_GETBAND` Returns the priority band of the first message on the stream head read queue in the integer referenced by *arg*. On failure, `errno` is set to the following value:

`ENODATA` No message on the stream head read queue.

`I_CANPUT` Check if a certain band is writable. *arg* is set to the priority band in question. The return value is `0` if the priority band *arg* is flow controlled, `1` if the band is writable, or `−1` on error. On failure, `errno` is set to the following value:

`EINVAL` Invalid *arg* value.

`I_SETCLTIME` Allows the user to set the time the stream head will delay when a stream is closing and there are data on the write queues. Before closing each module and driver, the stream head will delay for the specified amount of time to allow the data to drain. Note, however, that the module or driver may itself delay in its close routine; this delay is independent of the stream head's delay and is not settable. If, after the delay, data are still present, data will be flushed. *arg* is a pointer to an integer containing the number of milliseconds to delay, rounded up to the nearest legal value on the system. The default is fifteen seconds. On failure, `errno` is set to the following value:

`EINVAL` Invalid *arg* value.

`I_GETCLTIME` Returns the close time delay in the integer pointed by *arg*.

`I_SERROPT` Sets the error mode using the value of the argument *arg*.

Normally stream head errors are persistent; once they are set due to an `M_ERROR` or `M_HANGUP`, the error condition will remain until the stream is closed. This option can be used to set the stream head into non-persistent error mode i.e. once the error has been returned in response to a `read(2)`, `getmsg(2)`, `ioctl(2)`, `write(2)`, or `putmsg(2)` call the error condition will be cleared. The error mode can be controlled independently for read and write side errors. Legal *arg* values are either none or one of:

`RERRNORM` Persistent read errors, the default.

`RERRNONPERSIST` Non-persistent read errors.

OR'ed with either none or one of:

`WERRNORM` Persistent write errors, the default.

`WERRNONPERSIST` Non-persistent write errors.

When no value is specified e.g. for the read side error behavior then the behavior for that side will be left unchanged.

On failure, `errno` is set to the following value:

`EINVAL` *arg* is not one of the above legal values.

`I_GERROPT` Returns the current error mode setting in an `int` pointed to by the argument *arg*. Error modes are described above for `I_SERROPT`. On failure,`errno` is set to the following value:

`EFAULT` *arg* points outside the allocated address space.

The following four commands are used for connecting and disconnecting multiplexed STREAMS configurations.

`I_LINK` Connects two streams, where *fildes* is the file descriptor of the stream connected to the multiplexing driver, and *arg* is the file descriptor of the stream connected to another driver. The stream designated by *arg* gets connected below the multiplexing driver. `I_LINK` requires the multiplexing driver to send an acknowledgement message to the stream head regarding the linking operation. This call returns a multiplexor ID number (an identifier used to disconnect the multiplexor, see `I_UNLINK`) on success, and -1 on failure. On failure, `errno` is set to one of the following values:

`ENXIO` Hangup received on *fildes*.

`ETIME` Time out before acknowledgement message was received at stream head.

`EAGAIN` Temporarily unable to allocate storage to perform the `I_LINK`.

`ENOSR` Unable to allocate storage to perform the `I_LINK` due to insufficient STREAMS memory resources.

`EBADF` *arg* is not a valid, open file descriptor.

`EINVAL` *fildes* stream does not support multiplexing.

`EINVAL` *arg* is not a stream, or is already linked under a multiplexor.

`EINVAL` The specified link operation would cause a "cycle" in the resulting configuration; that is, a driver would be linked into the multiplexing configuration in more than one place.

`EINVAL` *fildes* is the file descriptor of a pipe or FIFO.

`EINVAL` Either the upper or lower stream has a major number >= the maximum major number on the system.

An `I_LINK` can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the stream head of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, `I_LINK` will fail with `errno` set to the value in the message.

`I_UNLINK` Disconnects the two streams specified by *fildes* and *arg*. *fildes* is the file descriptor of the stream connected to the multiplexing driver. *arg* is the multiplexor ID number that was returned by the `I_LINK`. If *arg* is -1, then all streams that were linked to *fildes* are disconnected. As in `I_LINK`, this command requires the multiplexing driver to acknowledge the unlink. On failure, `errno` is set to one of the following values:

`ENXIO` Hangup received on *fildes*.

ETIME Time out before acknowledgement message was received at stream head.

ENOSR Unable to allocate storage to perform the I_UNLINK due to insufficient STREAMS memory resources.

EINVAL *arg* is an invalid multiplexor ID number or *fildes* is not the stream on which the I_LINK that returned *arg* was performed.

EINVAL *fildes* is the file descriptor of a pipe or FIFO.

An I_UNLINK can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the stream head of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, I_UNLINK will fail with errno set to the value in the message.

I_PLINK Connects two streams, where *fildes* is the file descriptor of the stream connected to the multiplexing driver, and *arg* is the file descriptor of the stream connected to another driver. The stream designated by *arg* gets connected via a persistent link below the multiplexing driver. I_PLINK requires the multiplexing driver to send an acknowledgement message to the stream head regarding the linking operation. This call creates a persistent link that continues to exist even if the file descriptor *fildes* associated with the upper stream to the multiplexing driver is closed. This call returns a multiplexor ID number (an identifier that may be used to disconnect the multiplexor, see I_PUNLINK) on success, and -1 on failure. On failure, errno is set to one of the following values:

ENXIO Hangup received on *fildes*.

ETIME Time out before acknowledgement message was received at the stream head.

EAGAIN Unable to allocate STREAMS storage to perform the I_PLINK.

EBADF *arg* is not a valid, open file descriptor.

EINVAL *fildes* does not support multiplexing.

EINVAL *arg* is not a stream or is already linked under a multiplexor.

EINVAL The specified link operation would cause a "cycle" in the resulting configuration; that is, if a driver would be linked into the multiplexing configuration in more than one place.

EINVAL *fildes* is the file descriptor of a pipe or FIFO.

An I_PLINK can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error on a hangup is

received at the stream head of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, `I_PLINK` will fail with `errno` set to the value in the message.

`I_PUNLINK` Disconnects the two streams specified by *fildes* and *arg* that are connected with a persistent link. *fildes* is the file descriptor of the stream connected to the multiplexing driver. *arg* is the multiplexor ID number that was returned by `I_PLINK` when a stream was linked below the multiplexing driver. If *arg* is `MUXID_ALL` then all streams that are persistent links to *fildes* are disconnected. As in `I_PLINK`, this command requires the multiplexing driver to acknowledge the unlink. On failure, `errno` is set to one of the following values:

`ENXIO` Hangup received on *fildes*.

`ETIME` Time out before acknowledgement message was received at the stream head.

`EAGAIN` Unable to allocate buffers for the acknowledgement message.

`EINVAL` Invalid multiplexor ID number.

`EINVAL` *fildes* is the file descriptor of a pipe or FIFO.

An `I_PUNLINK` can also fail while waiting for the multiplexing driver to acknowledge the link request if a message indicating an error or a hangup is received at the stream head of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, `I_PUNLINK` will fail with `errno` set to the value in the message.

Return Values Unless specified otherwise above, the return value from `ioctl()` is `0` upon success and `−1` upon failure, with `errno` set as indicated.

See Also `Intro(3)`, `close(2)`, `fcntl(2)`, `getmsg(2)`, `ioctl(2)`, `open(2)`, `poll(2)`, `putmsg(2)`, `read(2)`, `write(2)`, `signal(3C)`, `signal.h(3HEAD)`,

STREAMS Programming Guide

Name su – asynchronous serial port driver

Synopsis

```
#include <fcntl.h>
#include <sys/termios.h>
open("/dev/tty[a-z]", _mode);
open("/dev/term[a-z]", _mode);
open("/dev/cua[a-z]", _mode);
```

Description The su module is a loadable STREAMS driver that provides basic support for standard UARTS that use Intel-8250, National Semiconductor-16450/16550 hardware and Southbridge 1535D (16550 compatable) Super I/O hardware. The module also provides keyboard and mouse I/O support for Sun machines using those same Intel, National Semiconductor and Southbridge chipsets. The su driver provides basic asynchronous communication support for serial ports. Both the serial devices and keyboard/mouse devices will have streams built with appropriate modules pushed atop the su driver by means of either the autopush(1M) or dacf.conf(4) facilities, depending on the OS revision and architecture in use.

The su module supports those termio(7I) device control functions specified by flags in the c_cflag word of the termios structure, and by the IGNBRK, IGNPAR, PARMRK, or INPCK flags in the c_iflag word of the termios structure. All other termio(7I) functions must be performed by STREAMS modules pushed atop the driver. When a device is opened, the ldterm(7M) and ttcompat(7M) STREAMS modules are automatically pushed on top of the stream, providing the standard termio(7I) interface.

The character-special devices /dev/ttya and /dev/ttyb are used to access the two standard serial ports. The su module supports up to ten serial ports, including the standard ports. The tty*[a-z]* devices have minor device numbers in the range 00-03, and may be assigned names of the form /dev/ttyd_n_, where _n_ denotes the line to be accessed. These device names are typically used to provide a logical access point for a _*dial-in*_ line that is used with a modem.

To allow a single tty line to be connected to a modem and used for incoming and outgoing calls, a special feature is available that is controlled by the minor device number. By accessing character-special devices with names of the form /dev/cua_*n*, it is possible to open a port without the Carrier Detect signal being asserted, either through hardware or an equivalent software mechanism. These devices are commonly known as _*dial-out*_ lines.

Application Programming Interface Once a /dev/cua_*n*_ line is opened, the corresponding tty, or ttyd line cannot be opened until the /dev/cua_*n*_ line is closed. A blocking open will wait until the /dev/cua_*n*_ line is closed (which will drop Data Terminal Ready, after which Carrier Detect will usually drop as well) and carrier is detected again. A non-blocking open will return an error. If the /dev/ttyd_*n*_ line has been opened successfully (usually only when carrier is recognized on the modem), the corresponding /dev/cua_*n*_ line cannot be opened. This allows a modem to be attached to a device, (for example, /dev/ttyd0, which is renamed from /dev/tty00) and used for dial-in (by enabling the line for login in /etc/inittab) or dial-out (by tip(1) or uucp(1C)) as /dev/cua0 when no one is logged in on the line.

ioctls The standard set of `termio` `ioctl()` calls are supported by `su`.

Breaks can be generated by the `TCSBRK`, `TIOCSBRK`, and `TIOCCBRK` `ioctl()` calls.

The input and output line speeds may be set to any of the following baud rates: 0, 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600 or 115200. The speeds cannot be set independently; for example, when the output speed is set, the input speed is automatically set to the same speed.

When the `su` module is used to service the serial console port, it supports a BREAK condition that allows the system to enter the debugger or the monitor. The BREAK condition is generated by hardware and it is usually enabled by default.

A BREAK condition originating from erroneous electrical signals cannot be distinguished from one deliberately sent by remote DCE. The Alternate Break sequence can be used as a remedy against this. Due to a risk of incorrect sequence interpretation, SLIP and certain other binary protocols should not be run over the serial console port when Alternate Break sequence is in effect. Although PPP is a binary protocol, it is able to avoid these sequences using the ACCM feature in *RFC 1662*. For Solaris PPP 4.0, you do this by adding the following line to the `/etc/ppp/options` file (or other configuration files used for the connection; see `pppd(1M)` for details):

```
asyncmap  0x00002000
```

By default, the Alternate Break sequence is a three character sequence: carriage return, tilde and control-B (CR ~ CTRL-B), but may be changed by the driver. For more information on breaking (entering the debugger or monitor), see `kbd(1)` and `kb(7M)`.

Errors An `open()` will fail under the following conditions:

`ENXIO` The unit being opened does not exist.

`EBUSY` The dial-out device is being opened while the dial-in device is already open, or the dial-in device is being opened with a no-delay open and the dial-out device is already open.

`EBUSY` The unit has been marked as exclusive-use by another process with a `TIOCEXCL` `ioctl()` call.

Files

`/dev/cua/[a-z]`	dial-out tty lines
`/dev/term/[a-z]`	dial-in tty lines
`/dev/tty[a-z]`	binary compatibility package device names

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC

See Also strconf(1), kbd(1), tip(1),uucp(1C), autopush(1M), kstat(1M), pppd(1M), ioctl(2), open(2), termios(3C), dacf.conf(4), attributes(5), kb(7M), ldterm(7M), ttcompat(7M), termio(7I)

Diagnostics The su driver keeps track of various warning and error conditions using kstat counters. The output of the kstat su command provides kstat counters. The counters and their meaning follow:

silo overflow — The internal chip FIFO received more data than it could handle. This indicates that the Solaris operating environment was not servicing data interrupts fast enough possibly due to a system with too many interrupts or a data line with a data rate that is too high.

ring buffer overflow — The su module was unable to store data it removed from the chips internal FIFO into a software buffer. The user process is not reading data fast enough, possibly due to an overloaded system. If possible, the application should enable flow control (either CTSRTS or XONXOFF) to allow the driver to backpressure the remote system when the local buffers fill up.

Name sv – Storage Volume system call device

Description The `sv` driver allows standard system call access (see `Intro(2)`) to a disk device to be redirected into the StorageTek architecture software. This enables standard applications to use Sun StorageTek Availability Suite components such as Point-in-Time Copy and Remote Mirror software.

Files `kernel/drv/sv` SV control and administration driver.

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWspsvu
Interface Stability	Committed

See Also `iiadm(1M)`

Name symhisl – symhisl SCSI Host Bus Adapter Driver

Synopsis `scsi@unit-address`

Description The `symhisl` host bus adapter driver is a SCSA-compliant nexus driver that supports the LSI Logic SYM53C895A, SYM53C1010-33, and SYM53C1010-66 SCSI controller chips.

The `symhisl` driver supports the standard functions provided by the SCSA interface, including tagged and untagged queuing, Wide, Fast, Ultra, Ultra2, and Ultra3 SCSI, and auto request sense. The `symhisl` driver does not support linked commands.

Configuration You configure the `symhisl` driver by defining properties in the `symhisl.conf` file. Properties in the `symhisl.conf` file override the global SCSI settings. The driver supports the following user-modifiable properties:

`scsi-options`
`target<`*n*`>-scsi-options`
`scsi-reset-delay`
`scsi-watchdog-tick`
`scsi-initiator-id`
`symFlags`

`target<`*n*`>-scsi-options` overrides the `scsi-options` property value for `target<`*n*`>`. <*n*> can vary from hex `0` to `f`. `symhisl` supports the following `scsi-options`: SCSI_OPTIONS_DR, SCSI_OPTIONS_SYNC, SCSI_OPTIONS_FAST, SCSI_OPTIONS_ULTRA, SCSI_OPTIONS_ULTRA2, SCSI_OPTIONS_TAG, and SCSI_OPTIONS_WIDE.

SCSI_OPTIONS_PARITY is supported for the `scsi-options` setting only and disables host adapter parity checking.

After periodic interval `scsi-watchdog-tick` (seconds), the `symhisl` driver searches through all current and disconnected commands for timeouts.

`symFlags` is a driver-specific bit-mask you can use to enable or disable driver properties.

`bit 0` When set, the driver will not reset the SCSI bus at initialization. Certain CD-ROM, tape, and other devices will not work properly when this bit is set. The default state for this bit is cleared.

`bit 1` When set, the driver will not export the DMI ioctl interface. Set this bit only if you want to disable the ioctl interface for security reasons. The default state for this bit is cleared.

`bit 2` When set, the driver disables 64-bit addressing capability. When clear, the driver enables 64-bit addressing capability. The default state for this bit is cleared.

`bit 3` When set, the driver disables SCSI domain validation for all devices on any adapters controlled by the driver.

Refer to scsi_hba_attach(9F) for more information on driver configuration.

Examples Edit the file /kernel/drv/symhisl.conf and add the following line:

```
scsi-options=0x78;
```

This disables tagged queuing, Fast, Ultra, and Ultra2 SCSI and wide mode for all symhisl instances.

The following example disables an option for one specific symhisl instance (refer to driver.conf(4) and pci(4) for more details):

```
name="symhisl" parent="/pci@1f,4000"
    unit-address="3"
    target1-scsi-options=0x58
    scsi-options=0x178 scsi-initiator-id=6;
```

Note that the initiator ID can only be changed for symhisl adapters that do not use the LSI Logic Boot ROM Configuration Utility. For adapters that can use the LSI Logic Boot ROM Configuration Utility, scsi-initiator-id has no effect.

The example above sets scsi-options for target 1 to 0x58 and all other targets on this SCSI bus to 0x178.

The physical path name of the parent can be determined using the /devices tree or following the link of the logical device name:

```
# ls –l /dev/rdsk/c0t0d0s0
lrwxrwxrwx  1 root   root      45 May 16 10:08 /dev/rdsk/c0t0d0s0 ->
   . . / . . /devices/pci@1f,4000/scsi@3/sd@0,0:a,raw
```

In this case, the parent is /pci@1f,4000 and the unit-address is the number bound to the scsi@3 node.

scsi-options specified per target ID have the highest precedence, followed by scsi-options per device type. Global scsi-options (for all symhisl instances) per bus have the lowest precedence.

The system must be rebooted for the specified scsi-options to take effect.

Driver Capabilities The target driver sets capabilities in the symhisl driver to enable some driver features. The target driver can query and modify the following capabilities: disconnect, synchronous, wide-xfer, tagged-qing, and auto-rqsense. All other capabilities are query only.

By default, tagged-qing capabilities are disabled, while disconnect, synchronous, wide-xfer, auto-rqsense, and untagged-qing are enabled. These capabilities can only have binary values (0 or 1).

The target driver must enable `tagged-qing` explicitly. The `untagged-qing` capability is always enabled and its value cannot be modified.

If a conflict exists between the value of `scsi-options` and a capability, the value set in `scsi-options` prevails. Only `whom != 0` is supported in the `scsi_ifsetcap(9F)` call. Refer to `scsi_ifsetcap(9F)` and `scsi_ifgetcap(9F)` for details.

Files

`/kernel/drv/symhisl`	ELF kernel module
`/kernel/drv/symhisl.conf`	Configuration file

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	Limited to PCI-based systems with LSI Logic SYM53C895A, SYM53C1010-33, and SYM53C1010-66 SCSI I/O processors.

See Also

`prtconf(1M)`, `driver.conf(4)`, `pci(4)`, `attributes(5)`, `scsi_abort(9F)`, `scsi_hba_attach(9F)`, `scsi_ifgetcap(9F)`, `scsi_ifsetcap(9F)`, `scsi_reset(9F)`, `scsi_sync_pkt(9F)`, `scsi_transport(9F)`, `scsi_device(9S)`, `scsi_extended_sense(9S)`, `scsi_inquiry(9S)`, `scsi_pkt(9S)`

Writing Device Drivers

ANSI Small Computer System Interface-2 (SCSI-2),

LSI Logic Corporation, *SYM53C896 PCI-SCSI I/O Processor*

LSI Logic Corporation, *SYM53C895A PCI-SCSI I/O Processor*

LSI Logic Corporation, *SYM53C1010 PCI-SCSI I/O Processor*

Notes

The `symhisl` SYM53C895A and SYM53C896 (SYM21002 and SYM22910) hardware and software support Wide, Fast, SCSI Ultra, and Ultra2 synchronous speeds. SYM53C1010-33 and SYM53C1010-66 also support Ultra3 synchronous speeds. The maximum SCSI bandwidth for Ultra2 transfers is 80 Mbytes/sec and 160 Mbytes/sec for Ultra3.

Name sysmsg – system message routing to console devices

Synopsis `/dev/sysmsg`

Description The file `/dev/sysmsg` routes output to a variable set of console devices. Writes to `/dev/sysmsg` are always directed to the system console `/dev/console`, and are in addition directed to a set of auxiliary console devices managed by `consadm(1m)`.

Only root has permission to write to this device.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsr
Interface Stability	Stable

See Also `consadm(1m)`, `syslogd(1M)` , `attributes(5)`, `console(7D)`

Name systrace – DTrace system call tracing provider

Description The `systrace` driver implements the DTrace `syscall` dynamic tracing provider. The syscall provider performs dynamic instrumentation to offer probes that fire whenever a thread enters or returns from a kernel system call entry point.

The `systrace` driver is not a public interface and you access the instrumentation offered by this provider through DTrace. Refer to the *Solaris Dynamic Tracing Guide* for a description of the public documented interfaces available for the DTrace facility and the probes offered by the `syscall` provider.

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWdtrp
Interface Stability	Private

See Also `dtrace(1M)`, `attributes(5)`, `dtrace(7D)`

Solaris Dynamic Tracing Guide

Name tavor – InfiniHost MT23108/MT25208 InfiniBand (IB) Driver

Synopsis

```
PCI    pci15b3,5a44@pci-slot, pci15b3,5a45@pci-slot
PCI-E pci15b3,6278@pci-e-slot, pci15b3,6279@pci-e-slot
```

Description The tavor driver is an IB Architecture-compliant implementation of an HCA, which operates on the Mellanox MT23108 InfiniBand ASSP and the Mellanox MT25208 InfiniBand ASSP. These ASSP's support the link and physical layers of the InfiniBand specification, while the ASSP and the driver support the transport layer.

The tavor driver interfaces with the InfiniBand Transport Framework (IBTF) and provides an implementation of the Channel Interfaces that are defined by that framework. It also enables management applications and agents to access the IB fabric.

Files

/kernel/drv/tavor	32-bit ELF kernel module (x86 platform only).
/kernel/drv/amd64/tavor	64-bit ELF kernel module (x86 platform only).
/kernel/drv/sparcv9/tavor	64–bit ELF Kernel Module (SPARC platform only).
/kernel/drv/tavor.conf	Driver configuration file.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	PCI-based systems
Availability	SUNWtavor

See Also `driver.conf(4)`, `printers.conf(4)`, `attributes(5)`

Writing Device Drivers

Diagnostics In addition to being logged, the following messages may appear on the system console.

tavor : driver attached for maintenance mode only.	There was a failure in the boot process of the tavor ASSP and the only function that can be performed is to re-flash firmware on the ASSP.
driver failed to attach.	The ASSP could not boot into either operational (HCA) mode or into maintenance mode. The device in inoperable.
Unexpected port number in port state change event.	A port state change event occurred, but the port number in the message does not exist on this HCA. This

	message also indicates the port number that was in the port state change.
Tavor driver successfully detached.	The driver has been removed from the system, and the HCA is no longer available for transfer operations.
tavor"n": port "m" up.	A port up asynchronous event has occured. "*n*" represents the instance of the Tavor device number, and "*m*" represents the port number on the Tavor device.
tavor"n": port "m" down.	A port down aynchronous event has occurred.
Tavor: <command name> command failed.	A internal firmware command failed to execute.

Name tcp, TCP – Internet Transmission Control Protocol

Synopsis

```
#include <sys/socket.h>

#include <netinet/in.h>

s = socket(AF_INET, SOCK_STREAM, 0);

s = socket(AF_INET6, SOCK_STREAM, 0);

t = t_open("/dev/tcp", O_RDWR);

t = t_open("/dev/tcp6", O_RDWR);
```

Description TCP is the virtual circuit protocol of the Internet protocol family. It provides reliable, flow-controlled, in order, two-way transmission of data. It is a byte-stream protocol layered above the Internet Protocol (IP), or the Internet Protocol Version 6 (IPv6), the Internet protocol family's internetwork datagram delivery protocol.

Programs can access TCP using the socket interface as a `SOCK_STREAM` socket type, or using the Transport Level Interface (TLI) where it supports the connection-oriented (`T_COTS_ORD`) service type.

TCP uses IP's host-level addressing and adds its own per-host collection of "port addresses." The endpoints of a TCP connection are identified by the combination of an IP or IPv6 address and a TCP port number. Although other protocols, such as the User Datagram Protocol (UDP), may use the same host and port address format, the port space of these protocols is distinct. See `inet(7P)` and `inet6(7P)` for details on the common aspects of addressing in the Internet protocol family.

Sockets utilizing TCP are either "active" or "passive." Active sockets initiate connections to passive sockets. Both types of sockets must have their local IP or IPv6 address and TCP port number bound with the `bind(3SOCKET)` system call after the socket is created. By default, TCP sockets are active. A passive socket is created by calling the `listen(3SOCKET)` system call after binding the socket with `bind()`. This establishes a queueing parameter for the passive socket. After this, connections to the passive socket can be received with the `accept(3SOCKET)` system call. Active sockets use the `connect(3SOCKET)` call after binding to initiate connections.

By using the special value `INADDR_ANY` with IP, or the unspecified address (all zeroes) with IPv6, the local IP address can be left unspecified in the `bind()` call by either active or passive TCP sockets. This feature is usually used if the local address is either unknown or irrelevant. If left unspecified, the local IP or IPv6 address will be bound at connection time to the address of the network interface used to service the connection.

Note that no two TCP sockets can be bound to the same port unless the bound IP addresses are different. IPv4 `INADDR_ANY` and IPv6 unspecified addresses compare as equal to any IPv4 or IPv6 address. For example, if a socket is bound to `INADDR_ANY` or unspecified address and

port X, no other socket can bind to port X, regardless of the binding address. This special consideration of INADDR_ANY and unspecified address can be changed using the socket option SO_REUSEADDR. If SO_REUSEADDR is set on a socket doing a bind, IPv4 INADDR_ANY and IPv6 unspecified address do not compare as equal to any IP address. This means that as long as the two sockets are not both bound to INADDR_ANY/unspecified address or the same IP address, the two sockets can be bound to the same port.

If an application does not want to allow another socket using the SO_REUSEADDR option to bind to a port its socket is bound to, the application can set the socket level option SO_EXCLBIND on a socket. The option values of 0 and 1 mean enabling and disabling the option respectively. Once this option is enabled on a socket, no other socket can be bound to the same port.

Once a connection has been established, data can be exchanged using the read(2) and write(2) system calls.

Under most circumstances, TCP sends data when it is presented. When outstanding data has not yet been acknowledged, TCP gathers small amounts of output to be sent in a single packet once an acknowledgement has been received. For a small number of clients, such as window systems that send a stream of mouse events which receive no replies, this packetization may cause significant delays. To circumvent this problem, TCP provides a socket-level boolean option, TCP_NODELAY. TCP_NODELAY is defined in <netinet/tcp.h>, and is set with setsockopt(3SOCKET) and tested with getsockopt(3SOCKET). The option level for the setsockopt() call is the protocol number for TCP, available from getprotobyname(3SOCKET).

For some applications, it may be desirable for TCP not to send out data unless a full TCP segment can be sent. To enable this behavior, an application can use the TCP_CORK socket option. When TCP_CORK is set with a non-zero value, TCP sends out a full TCP segment only. When TCP_CORK is set to zero after it has been enabled, all buffered data is sent out (as permitted by the peer's receive window and the current congestion window). TCP_CORK is defined in <netinet/tcp.h>, and is set with setsockopt(3SOCKET) and tested with getsockopt(3SOCKET). The option level for the setsockopt() call is the protocol number for TCP, available from getprotobyname(3SOCKET).

The SO_RCVBUF socket level option can be used to control the window that TCP advertises to the peer. IP level options may also be used with TCP. See ip(7P) and ip6(7P).

Another socket level option, SO_RCVBUF, can be used to control the window that TCP advertises to the peer. IP level options may also be used with TCP. See ip(7P) and ip6(7P).

TCP provides an urgent data mechanism, which may be invoked using the out-of-band provisions of send(3SOCKET). The caller may mark one byte as “urgent” with the MSG_OOB flag to send(3SOCKET). This sets an “urgent pointer” pointing to this byte in the TCP stream. The receiver on the other side of the stream is notified of the urgent data by a SIGURG signal. The SIOCATMARK ioctl(2) request returns a value indicating whether the stream is at the

urgent mark. Because the system never returns data across the urgent mark in a single read(2) call, it is possible to advance to the urgent data in a simple loop which reads data, testing the socket with the SIOCATMARK ioctl() request, until it reaches the mark.

Incoming connection requests that include an IP source route option are noted, and the reverse source route is used in responding.

A checksum over all data helps TCP implement reliability. Using a window-based flow control mechanism that makes use of positive acknowledgements, sequence numbers, and a retransmission strategy, TCP can usually recover when datagrams are damaged, delayed, duplicated or delivered out of order by the underlying communication medium.

If the local TCP receives no acknowledgements from its peer for a period of time, (for example, if the remote machine crashes), the connection is closed and an error is returned.

TCP follows the congestion control algorithm described in *RFC 2581*, and also supports the initial congestion window (cwnd) changes in *RFC 3390*. The initial cwnd calculation can be overridden by the socket option TCP_INIT_CWND. An application can use this option to set the initial cwnd to a specified number of TCP segments. This applies to the cases when the connection first starts and restarts after an idle period. The process must have the PRIV_SYS_NET_CONFIG privilege if it wants to specify a number greater than that calculated by *RFC 3390*.

SunOS supports TCP Extensions for High Performance (*RFC 1323*) which includes the window scale and time stamp options, and Protection Against Wrap Around Sequence Numbers (PAWS). SunOS also supports Selective Acknowledgment (SACK) capabilities (RFC 2018) and Explicit Congestion Notification (ECN) mechanism (*RFC 3168*).

Turn on the window scale option in one of the following ways:

- An application can set SO_SNDBUF or SO_RCVBUF size in the setsockopt() option to be larger than 64K. This must be done *before* the program calls listen() or connect(), because the window scale option is negotiated when the connection is established. Once the connection has been made, it is too late to increase the send or receive window beyond the default TCP limit of 64K.
- For all applications, use ndd(1M) to modify the configuration parameter tcp_wscale_always. If tcp_wscale_always is set to 1, the window scale option will always be set when connecting to a remote system. If tcp_wscale_always is 0, the window scale option will be set only if the user has requested a send or receive window larger than 64K. The default value of tcp_wscale_always is 1.
- Regardless of the value of tcp_wscale_always, the window scale option will always be included in a connect acknowledgement if the connecting system has used the option.

Turn on SACK capabilities in the following way:

- Use ndd to modify the configuration parameter tcp_sack_permitted. If tcp_sack_permitted is set to 0, TCP will not accept SACK or send out SACK information. If tcp_sack_permitted is set to 1, TCP will not initiate a connection with SACK permitted option in the SYN segment, but will respond with SACK permitted option in the SYN|ACK segment if an incoming connection request has the SACK permitted option. This means that TCP will only accept SACK information if the other side of the connection also accepts SACK information. If tcp_sack_permitted is set to 2, it will both initiate and accept connections with SACK information. The default for tcp_sack_permitted is 2 (active enabled).

Turn on TCP ECN mechanism in the following way:

- Use ndd to modify the configuration parameter tcp_ecn_permitted. If tcp_ecn_permitted is set to 0, TCP will not negotiate with a peer that supports ECN mechanism. If tcp_ecn_permitted is set to 1 when initiating a connection, TCP will not tell a peer that it supports ECN mechanism. However, it will tell a peer that it supports ECN mechanism when accepting a new incoming connection request if the peer indicates that it supports ECN mechanism in the SYN segment. If tcp_ecn_permitted is set to 2, in addition to negotiating with a peer on ECN mechanism when accepting connections, TCP will indicate in the outgoing SYN segment that it supports ECN mechanism when TCP makes active outgoing connections. The default for tcp_ecn_permitted is 1.

Turn on the time stamp option in the following way:

- Use ndd to modify the configuration parameter tcp_tstamp_always. If tcp_tstamp_always is 1, the time stamp option will always be set when connecting to a remote machine. If tcp_tstamp_always is 0, the timestamp option will not be set when connecting to a remote system. The default for tcp_tstamp_always is 0.
- Regardless of the value of tcp_tstamp_always, the time stamp option will always be included in a connect acknowledgement (and all succeeding packets) if the connecting system has used the time stamp option.

Use the following procedure to turn on the time stamp option only when the window scale option is in effect:

- Use ndd to modify the configuration parameter tcp_tstamp_if_wscale. Setting tcp_tstamp_if_wscale to 1 will cause the time stamp option to be set when connecting to a remote system, if the window scale option has been set. If tcp_tstamp_if_wscale is 0, the time stamp option will not be set when connecting to a remote system. The default for tcp_tstamp_if_wscale is 1.

Protection Against Wrap Around Sequence Numbers (PAWS) is always used when the time stamp option is set.

SunOS also supports multiple methods of generating initial sequence numbers. One of these methods is the improved technique suggested in RFC 1948. We *HIGHLY* recommend that you

set sequence number generation parameters as close to boot time as possible. This prevents sequence number problems on connections that use the same connection-ID as ones that used a different sequence number generation. The `svc:/network/initial:default` service configures the initial sequence number generation. The service reads the value contained in the configuration file `/etc/default/inetinit` to determine which method to use.

The `/etc/default/inetinit` file is an unstable interface, and may change in future releases.

TCP may be configured to report some information on connections that terminate by means of an RST packet. By default, no logging is done. If the `ndd(1M)` parameter *tcp_trace* is set to 1, then trace data is collected for all new connections established after that time.

The trace data consists of the TCP headers and IP source and destination addresses of the last few packets sent in each direction before RST occurred. Those packets are logged in a series of `strlog(9F)` calls. This trace facility has a very low overhead, and so is superior to such utilities as `snoop(1M)` for non-intrusive debugging for connections terminating by means of an RST.

SunOS supports the keep-alive mechanism described in *RFC 1122*. It is enabled using the socket option SO_KEEPALIVE. When enabled, the first keep-alive probe is sent out after a TCP is idle for two hours If the peer does not respond to the probe within eight minutes, the TCP connection is aborted. You can alter the interval for sending out the first probe using the socket option TCP_KEEPALIVE_THRESHOLD. The option value is an unsigned integer in milliseconds. The system default is controlled by the TCP ndd parameter tcp_keepalive_interval. The minimum value is ten seconds. The maximum is ten days, while the default is two hours. If you receive no response to the probe, you can use the TCP_KEEPALIVE_ABORT_THRESHOLD socket option to change the time threshold for aborting a TCP connection. The option value is an unsigned integer in milliseconds. The value zero indicates that TCP should never time out and abort the connection when probing. The system default is controlled by the TCP ndd parameter tcp_keepalive_abort_interval. The default is eight minutes.

See Also `svcs(1)`, `ndd(1M)`, `ioctl(2)`, `read(2)`, `svcadm(1M)`, `write(2)`, `accept(3SOCKET)`, `bind(3SOCKET)`, `connect(3SOCKET)`, `getprotobyname(3SOCKET)`, `getsockopt(3SOCKET)`, `listen(3SOCKET)`, `send(3SOCKET)`, `smf(5)`, `inet(7P)`, `inet6(7P)`, `ip(7P)`, `ip6(7P)`

Ramakrishnan, K., Floyd, S., Black, D., RFC 3168, *The Addition of Explicit Congestion Notification (ECN) to IP*, September 2001.

Mathias, M. and Hahdavi, J. Pittsburgh Supercomputing Center; Ford, S. Lawrence Berkeley National Laboratory; Romanow, A. Sun Microsystems, Inc. *RFC 2018, TCP Selective Acknowledgement Options*, October 1996.

Bellovin, S., *RFC 1948, Defending Against Sequence Number Attacks*, May 1996.

Jacobson, V., Braden, R., and Borman, D., *RFC 1323, TCP Extensions for High Performance*, May 1992.

Postel, Jon, *RFC 793, Transmission Control Protocol - DARPA Internet Program Protocol Specification*, Network Information Center, SRI International, Menlo Park, CA., September 1981.

Diagnostics A socket operation may fail if:

`EISCONN`	A `connect()` operation was attempted on a socket on which a `connect()` operation had already been performed.
`ETIMEDOUT`	A connection was dropped due to excessive retransmissions.
`ECONNRESET`	The remote peer forced the connection to be closed (usually because the remote machine has lost state information about the connection due to a crash).
`ECONNREFUSED`	The remote peer actively refused connection establishment (usually because no process is listening to the port).
`EADDRINUSE`	A `bind()` operation was attempted on a socket with a network address/port pair that has already been bound to another socket.
`EADDRNOTAVAIL`	A `bind()` operation was attempted on a socket with a network address for which no network interface exists.
`EACCES`	A `bind()` operation was attempted with a "reserved" port number and the effective user ID of the process was not the privileged user.
`ENOBUFS`	The system ran out of memory for internal data structures.

Notes The `tcp` service is managed by the service management facility, `smf(5)`, under the service identifier:

```
svc:/network/initial:default
```

Administrative actions on this service, such as enabling, disabling, or requesting restart, can be performed using `svcadm(1M)`. The service's status can be queried using the `svcs(1)` command.

Name termio – general terminal interface

Synopsis

```
#include <termio.h>

ioctl(int fildes, int request, struct termio *arg);

ioctl(int fildes, int request, int arg);

#include <termios.h>

ioctl(int fildes, int request, struct termios *arg);
```

Description This release supports a general interface for asynchronous communications ports that is hardware-independent. The user interface to this functionality is using function calls (the preferred interface) described in `termios(3C)` or `ioctl` commands described in this section. This section also discusses the common features of the terminal subsystem which are relevant with both user interfaces.

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, user programs seldom open terminal files; they are opened by the system and become a user's standard input, output, and error files. The first terminal file opened by the session leader that is not already associated with a session becomes the controlling terminal for that session. The controlling terminal plays a special role in handling quit and interrupt signals, as discussed below. The controlling terminal is inherited by a child process during a `fork(2)`. A process can break this association by changing its session using `setsid()` (see `setsid(2)`).

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters may be typed at any time, even while output is occurring, and are only lost when the character input buffers of the system become completely full, which is rare. For example, the number of characters in the line discipline buffer may exceed {`MAX_CANON`} and `IMAXBEL` (see below) is not set, or the user may accumulate {`MAX_INPUT`} number of input characters that have not yet been read by some program. When the input limit is reached, all the characters saved in the buffer up to that point are thrown away without notice.

Session Management (Job Control) A control terminal will distinguish one of the process groups in the session associated with it to be the foreground process group. All other process groups in the session are designated as background process groups. This foreground process group plays a special role in handling signal-generating input characters, as discussed below. By default, when a controlling terminal is allocated, the controlling process's process group is assigned as foreground process group.

Background process groups in the controlling process's session are subject to a job control line discipline when they attempt to access their controlling terminal. Process groups can be sent signals that will cause them to stop, unless they have made other arrangements. An exception is made for members of orphaned process groups.

An orphaned process group is one where the process group (see getpgid(2)) has no members with a parent in a different process group but sharing the same controlling terminal. When a member of an orphaned process group attempts to access its controlling terminal, EIO is returned because there would be no way to restart the process if it were stopped on one of these signals.

If a member of a background process group attempts to read its controlling terminal, its process group will be sent a SIGTTIN signal, which will normally cause the members of that process group to stop. If, however, the process is ignoring or holding SIGTTIN, or is a member of an orphaned process group, the read will fail with errno set to EIO, and no signal is sent.

If a member of a background process group attempts to write its controlling terminal and the TOSTOP bit is set in the c_lflag field, its process group is sent a SIGTTOU signal, which will normally cause the members of that process group to stop. If, however, the process is ignoring or holding SIGTTOU, the write will succeed. If the process is not ignoring or holding SIGTTOU and is a member of an orphaned process group, the write will fail with errno set to EIO, and no signal will be sent.

If TOSTOP is set and a member of a background process group attempts to ioctl its controlling terminal, and that ioctl will modify terminal parameters (for example, TCSETA, TCSETAW, TCSETAF, or TIOCSPGRP), its process group will be sent a SIGTTOU signal, which will normally cause the members of that process group to stop. If, however, the process is ignoring or holding SIGTTOU, the ioctl will succeed. If the process is not ignoring or holding SIGTTOU and is a member of an orphaned process group, the write will fail with errno set to EIO, and no signal will be sent.

Canonical Mode Input Processing

Normally, terminal input is processed in units of lines. A line is delimited by a newline (ASCII LF) character, an end-of-file (ASCII EOT) character, or an end-of-line character. This means that a program attempting to read will block until an entire line has been typed. Also, no matter how many characters are requested in the read call, at most one line will be returned. It is not necessary, however, to read a whole line at once; any number of characters may be requested in a read, even one, without losing information.

During input, erase and kill processing is normally done. The ERASE character (by default, the character DEL) erases the last character typed. The WERASE character (the character Control-w) erases the last "word" typed in the current input line (but not any preceding spaces or tabs). A "word" is defined as a sequence of non-blank characters, with tabs counted as blanks. Neither ERASE nor WERASE will erase beyond the beginning of the line. The KILL character (by default, the character NAK) kills (deletes) the entire input line, and optionally outputs a newline character. All these characters operate on a key stroke basis, independent of any backspacing or tabbing that may have been done. The REPRINT character (the character Control-r) prints a newline followed by all characters that have not been read. Reprinting also occurs automatically if characters that would normally be erased from the screen are fouled by program output. The characters are reprinted as if they were being echoed; consequencely, if ECHO is not set, they are not printed.

The ERASE and KILL characters may be entered literally by preceding them with the escape character. In this case, the escape character is not read. The erase and kill characters may be changed.

Non-canonical Mode Input Processing

In non-canonical mode input processing, input characters are not assembled into lines, and erase and kill processing does not occur. The MIN and TIME values are used to determine how to process the characters received.

MIN represents the minimum number of characters that should be received when the read is satisfied (that is, when the characters are returned to the user). TIME is a timer of 0.10-second granularity that is used to timeout bursty and short-term data transmissions. The four possible values for MIN and TIME and their interactions are described below.

Case A: MIN > 0, TIME > 0

In this case, TIME serves as an intercharacter timer and is activated after the first character is received. Since it is an intercharacter timer, it is reset after a character is received. The interaction between MIN and TIME is as follows: as soon as one character is received, the intercharacter timer is started. If MIN characters are received before the intercharacter timer expires (note that the timer is reset upon receipt of each character), the read is satisfied. If the timer expires before MIN characters are received, the characters received to that point are returned to the user. Note that if TIME expires, at least one character will be returned because the timer would not have been enabled unless a character was received. In this case (MIN > 0, TIME > 0), the read sleeps until the MIN and TIME mechanisms are activated by the receipt of the first character. If the number of characters read is less than the number of characters available, the timer is not reactivated and the subsequent read is satisfied immediately.

Case B: MIN > 0, TIME = 0

In this case, since the value of TIME is zero, the timer plays no role and only MIN is significant. A pending read is not satisfied until MIN characters are received (the pending read sleeps until MIN characters are received). A program that uses this case to read record based terminal I/O may block indefinitely in the read operation.

Case C: MIN = 0, TIME > 0

In this case, since MIN = 0, TIME no longer represents an intercharacter timer: it now serves as a read timer that is activated as soon as a `read` is done. A read is satisfied as soon as a single character is received or the read timer expires. Note that, in this case, if the timer expires, no character is returned. If the timer does not expire, the only way the read can be satisfied is if a character is received. In this case, the read will

not block indefinitely waiting for a character; if no character is received within TIME *.10 seconds after the read is initiated, the read returns with zero characters.

Case D: MIN = 0, TIME = 0 — In this case, return is immediate. The minimum of either the number of characters requested or the number of characters currently available is returned without waiting for more characters to be input.

Comparing Different Cases of MIN, TIME Interaction

Some points to note about MIN and TIME :

- In the following explanations, note that the interactions of MIN and TIME are not symmetric. For example, when MIN > 0 and TIME = 0, TIME has no effect. However, in the opposite case, where MIN = 0 and TIME > 0, both MIN and TIME play a role in that MIN is satisfied with the receipt of a single character.
- Also note that in case A (MIN > 0, TIME > 0), TIME represents an intercharacter timer, whereas in case C (MIN = 0, TIME > 0), TIME represents a read timer.

These two points highlight the dual purpose of the MIN/TIME feature. Cases A and B, where MIN > 0, exist to handle burst mode activity (for example, file transfer programs), where a program would like to process at least MIN characters at a time. In case A, the intercharacter timer is activated by a user as a safety measure; in case B, the timer is turned off.

Cases C and D exist to handle single character, timed transfers. These cases are readily adaptable to screen-based applications that need to know if a character is present in the input queue before refreshing the screen. In case C, the read is timed, whereas in case D, it is not.

Another important note is that MIN is always just a minimum. It does not denote a record length. For example, if a program does a read of 20 bytes, MIN is 10, and 25 characters are present, then 20 characters will be returned to the user.

Writing Characters

When one or more characters are written, they are transmitted to the terminal as soon as previously written characters have finished typing. Input characters are echoed as they are typed if echoing has been enabled. If a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue is drained down to some threshold, the program is resumed.

Special Characters

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

INTR — (Control-c or ASCII ETX) generates a SIGINT signal. SIGINT is sent to all foreground processes associated with the controlling terminal. Normally, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed upon location. (See `signal.h`(3HEAD)).

QUIT	(Control-| or ASCII FS) generates a SIGQUIT signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated but a core image file (called `core`) will be created in the current working directory.
ERASE	(DEL) erases the preceding character. It does not erase beyond the start of a line, as delimited by a NL, EOF, EOL, or EOL2 character.
WERASE	(Control-w or ASCII ETX) erases the preceding "word". It does not erase beyond the start of a line, as delimited by a NL, EOF, EOL, or EOL2 character.
KILL	(Control-u or ASCII NAK) deletes the entire line, as delimited by a NL, EOF, EOL, or EOL2 character.
REPRINT	(Control-r or ASCII DC2) reprints all characters, preceded by a newline, that have not been read.
EOF	(Control-d or ASCII EOT) may be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a newline, and the EOF is discarded. Thus, if no characters are waiting (that is, the EOF occurred at the beginning of a line) zero characters are passed back, which is the standard end-of-file indication. Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.
NL	(ASCII LF) is the normal line delimiter. It cannot be changed or escaped.
EOL	(ASCII NULL) is an additional line delimiter, like NL . It is not normally used.
EOL2	is another additional line delimiter.
SWTCH	(Control-z or ASCII EM) Header file symbols related to this special character are present for compatibility purposes only and the kernel takes no special action on matching SWTCH (except to discard the character).
SUSP	(Control-z or ASCII SUB) generates a SIGTSTP signal. SIGTSTP stops all processes in the foreground process group for that terminal.
DSUSP	(Control-y or ASCII EM). It generates a SIGTSTP signal as SUSP does, but the signal is sent when a process in the foreground process group attempts to read the DSUSP character, rather than when it is typed.
STOP	(Control-s or ASCII DC3) can be used to suspend output temporarily. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.
START	(Control-q or ASCII DC1) is used to resume output. Output has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read.

DISCARD (Control-o or ASCII SI) causes subsequent output to be discarded. Output is discarded until another DISCARD character is typed, more input arrives, or the condition is cleared by a program.

LNEXT (Control-v or ASCII SYN) causes the special meaning of the next character to be ignored. This works for all the special characters mentioned above. It allows characters to be input that would otherwise be interpreted by the system (for example KILL, QUIT). The character values for INTR, QUIT, ERASE, WERASE, KILL, REPRINT, EOF, EOL, EOL2, SWTCH, SUSP, DSUSP, STOP, START, DISCARD, and LNEXT may be changed to suit individual tastes. If the value of a special control character is _POSIX_VDISABLE (0), the function of that special control character is disabled. The ERASE, KILL, and EOF characters may be escaped by a preceding backslash (\) character, in which case no special function is done. Any of the special characters may be preceded by the LNEXT character, in which case no special function is done.

Modem Disconnect

When a modem disconnect is detected, a `SIGHUP` signal is sent to the terminal's controlling process. Unless other arrangements have been made, these signals cause the process to terminate. If `SIGHUP` is ignored or caught, any subsequent read returns with an end-of-file indication until the terminal is closed.

If the controlling process is not in the foreground process group of the terminal, a `SIGTSTP` is sent to the terminal's foreground process group. Unless other arrangements have been made, these signals cause the processes to stop.

Processes in background process groups that attempt to access the controlling terminal after modem disconnect while the terminal is still allocated to the session will receive appropriate `SIGTTOU` and `SIGTTIN` signals. Unless other arrangements have been made, this signal causes the processes to stop.

The controlling terminal will remain in this state until it is reinitialized with a successful open by the controlling process, or deallocated by the controlling process.

Terminal Parameters

The parameters that control the behavior of devices and modules providing the `termios` interface are specified by the `termios` structure defined by `termios.h`. Several `ioctl(2)` system calls that fetch or change these parameters use this structure that contains the following members:

```
tcflag_t c_iflag;  /* input modes */
   tcflag_t    c_oflag;  /* output modes */
   tcflag_t    c_cflag;  /* control modes */
   tcflag_t    c_lflag;  /* local modes */
   cc_t  c_cc[NCCS];  /* control chars */
```

The special control characters are defined by the array `c_cc`. The symbolic name `NCCS` is the size of the Control-character array and is also defined by `<termios.h>`. The relative positions, subscript names, and typical default values for each function are as follows:

Relative Position	Subscript Name	Typical Default Value
0	VINTR	ETX
1	VQUIT	FS
2	VERASE	DEL
3	VKILL	NAK
4	VEOF	EOT
5	VEOL	NUL
6	VEOL2	NUL
7	VWSTCH	NUL
8	VSTART	NUL
9	VSTOP	DC3
10	VSUSP	SUB
11	VDSUSP	EM
12	VREPRINT	DC2
13	VDISCARD	SI
14	VWERASE	ETB
15	VLNEXT	SYN
16-19	Reserved	

Input Modes

The `c_iflag` field describes the basic terminal input control:

`IGNBRK` Ignore break condition.

`BRKINT` Signal interrupt on break.

`IGNPAR` Ignore characters with parity errors.

`PARMRK` Mark parity errors.

`INPCK` Enable input parity check.

`ISTRIP` Strip character.

`INLCR` Map NL to CR on input.

IGNCR	Ignore CR.
ICRNL	Map CR to NL on input.
IUCLC	Map upper-case to lower-case on input.
IXON	Enable start/stop output control.
IXANY	Enable any character to restart output.
IXOFF	Enable start/stop input control.
IMAXBEL	Echo BEL on input line too long.

If IGNBRK is set, a break condition (a character framing error with data all zeros) detected on input is ignored, that is, not put on the input queue and therefore not read by any process. If IGNBRK is not set and BRKINT is set, the break condition shall flush the input and output queues and if the terminal is the controlling terminal of a foreground process group, the break condition generates a single SIGINT signal to that foreground process group. If neither IGNBRK nor BRKINT is set, a break condition is read as a single '\0' (ASCII NULL) character, or if PARMRK is set, as '\377', '\0', c, where '\377' is a single character with value 377 octal (0xff hex, 255 decimal), '\0' is a single character with value 0, and c is the errored character received.

If IGNPAR is set, a byte with framing or parity errors (other than break) is ignored.

If PARMRK is set, and IGNPAR is not set, a byte with a framing or parity error (other than break) is given to the application as the three-character sequence: '\377', '\0', c, where '\377' is a single character with value 377 octal (0xff hex, 255 decimal), '\0' is a single character with value 0, and c is the errored character received. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of '\377' is given to the application as '\377.' If neither IGNPAR nor PARMRK is set, a framing or parity error (other than break) is given to the application as a single '\0' (ASCII NULL) character.

If INPCK is set, input parity checking is enabled. If INPCK is not set, input parity checking is disabled. This allows output parity generation without input parity errors. Note that whether input parity checking is enabled or disabled is independent of whether parity detection is enabled or disabled. If parity detection is enabled but input parity checking is disabled, the hardware to which the terminal is connected will recognize the parity bit, but the terminal special file will not check whether this is set correctly or not.

If ISTRIP is set, valid input characters are first stripped to seven bits, otherwise all eight bits are processed.

If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, a received CR character is ignored (not read). Otherwise, if ICRNL is set, a received CR character is translated into a NL character.

If IUCLC is set, a received upper case, alphabetic character is translated into the corresponding lower case character.

If IXON is set, start/stop output control is enabled. A received STOP character suspends output and a received START character restarts output. The STOP and START characters will not be read, but will merely perform flow control functions. If IXANY is set, any input character restarts output that has been suspended.

If IXOFF is set, the system transmits a STOP character when the input queue is nearly full, and a START character when enough input has been read so that the input queue is nearly empty again.

If IMAXBEL is set, the ASCII BEL character is echoed if the input stream overflows. Further input is not stored, but any input already present in the input stream is not disturbed. If IMAXBEL is not set, no BEL character is echoed, and all input present in the input queue is discarded if the input stream overflows.

Output Modes

The c_oflag field specifies the system treatment of output:

OPOST Post-process output.

OLCUC Map lower case to upper on output.

ONLCR Map NL to CR-NL on output.

OCRNL Map CR to NL on output.

ONOCR No CR output at column 0.

ONLRET NL performs CR function.

OFILL Use fill characters for delay.

OFDEL Fill is DEL, else NULL.

NLDLY Select newline delays:

NL0
NL1

CRDLY Select carriage-return delays:

CR0
CR1
CR2
CR3

TABDLY Select horizontal tab delays or tab expansion:

TAB0

TAB1

TAB2

TAB3 Expand tabs to spaces

XTABS Expand tabs to spaces

BSDLY Select backspace delays:

BS0
BS1

VTDLY Select vertical tab delays:

VT0
VT1

FFDLY Select form feed delays:

FF0
FF1

If OPOST is set, output characters are post-processed as indicated by the remaining flags; otherwise, characters are transmitted without change.

If OLCUC is set, a lower case alphabetic character is transmitted as the corresponding upper case character. This function is often used in conjunction with IUCLC .

If ONLCR is set, the NL character is transmitted as the CR-NL character pair. If OCRNL is set, the CR character is transmitted as the NL character. If ONOCR is set, no CR character is transmitted when at column 0 (first position). If ONRET is set, the NL character is assumed to do the carriage-return function; the column pointer is set to 0 and the delays specified for CR are used. Otherwise, the NL character is assumed to do just the line-feed function; the column pointer remains unchanged. The column pointer is also set to 0 if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases, a value of 0 indicates no delay. If OFILL is set, fill characters are transmitted for delay instead of a timed delay. This is useful for high baud rate terminals that need only a minimal delay. If OFDEL is set, the fill character is DEL ; otherwise it is NULL.

If a form-feed or vertical-tab delay is specified, it lasts for about 2 seconds.

Newline delay lasts about 0.10 seconds. If `ONLRET` is set, the carriage-return delays are used instead of the newline delays. If `OFILL` is set, two fill characters are transmitted.

Carriage-return delay type 1 is dependent on the current column position, type 2 is about 0.10 seconds, and type 3 is about 0.15 seconds. If `OFILL` is set, delay type 1 transmits two fill characters, and type 2 transmits four fill characters.

Horizontal-tab delay type 1 is dependent on the current column position. Type 2 is about 0.10 seconds. Type 3 specifies that tabs are to be expanded into spaces. If `OFILL` is set, two fill characters are transmitted for any delay.

Backspace delay lasts about 0.05 seconds. If `OFILL` is set, one fill character is transmitted.

The actual delays depend on line speed and system load.

Control Modes

The `c_cflag` field describes the hardware control of the terminal:

`CBAUD`	Baud rate:
`B0`	Hang up
`B50`	50 baud
`B75`	75 baud
`B110`	110 baud
`B134`	134 baud
`B150`	150 baud
`B200`	200 baud
`B300`	300 baud
`B600`	600 baud
`B1200`	1200 baud
`B1800`	1800 baud
`B2400`	2400 baud
`B4800`	4800 baud
`B9600`	9600 baud
`B19200`	19200 baud
`EXTA`	External A
`B38400`	38400 baud
`EXTB`	External B

B57600	57600 baud
B76800	76800 baud
B115200	115200 baud
B153600	153600 baud
B230400	230400 baud
B307200	307200 baud
B460800	460800 baud
CSIZE	Character size:
CS5	5 bits
CS6	6 bits
CS7	7 bits
CS8	8 bits
CSTOPB	Send two stop bits, else one
CREAD	Enable receiver
PARENB	Parity enable
PARODD	Odd parity, else even
HUPCL	Hang up on last close
CLOCAL	Local line, else dial-up
CIBAUD	Input baud rate, if different from output rate
PAREXT	Extended parity for mark and space parity
CRTSXOFF	Enable inbound hardware flow control
CRTSCTS	Enable outbound hardware flow control
CBAUDEXT	Bit to indicate output speed > B38400
CIBAUDEXT	Bit to indicate input speed > B38400

The CBAUD bits together with the CBAUDEXT bit specify the output baud rate. To retrieve the output speed from the termios structure pointed to by termios_p see the following code segment.

```
speed_t ospeed;
if (termios_p->c_cflag & CBAUDEXT)
   ospeed = (termios_p->c_cflag & CBAUD) + CBAUD + 1;
```

```
else
   ospeed = termios_p->c_cflag & CBAUD;
```

To store the output speed in the termios structure pointed to by `termios_p` see the following code segment.

```
speed_t ospeed;
if (ospeed > CBAUD) {
   termios_p->c_cflag |= CBAUDEXT;
   ospeed -= (CBAUD + 1);
} else
   termios_p->c_cflag &= ~CBAUDEXT;
 termios_p->c_cflag =
    (termios_p->c_cflag & ~CBAUD) | (ospeed & CBAUD);
```

The zero baud rate, B0, is used to hang up the connection. If B0 is specified, the data-terminal-ready signal is not asserted. Normally, this disconnects the line.

If the `CIBAUDEXT` or `CIBAUD` bits are not zero, they specify the input baud rate, with the `CBAUDEXT` and `CBAUD` bits specifying the output baud rate; otherwise, the output and input baud rates are both specified by the `CBAUDEXT` and `CBAUD` bits. The values for the `CIBAUD` bits are the same as the values for the `CBAUD` bits, shifted left `IBSHIFT` bits. For any particular hardware, impossible speed changes are ignored. To retrieve the input speed in the `termios` structure pointed to by `termios_p` see the following code segment.

```
speed_t ispeed;
if (termios_p->c_cflag & CIBAUDEXT)
   ispeed = ((termios_p->c_cflag & CIBAUD) >> IBSHIFT)
   + (CIBAUD >> IBSHIFT) + 1;
else
   ispeed = (termios_p->c_cflag & CIBAUD) >> IBSHIFT;
```

To store the input speed in the `termios` structure pointed to by `termios_p` see the following code segment.

```
speed_t ispeed;
if (ispeed == 0) {
   ispeed = termios_p->c_cflag & CBAUD;
if (termios_p->c_cflag & CBAUDEXT)
   ispeed += (CBAUD + 1);
}
 if ((ispeed << IBSHIFT) > CIBAUD) {
    termios_p->c_cflag |= CIBAUDEXT;
    ispeed -= ((CIBAUD >> IBSHIFT) + 1);
} else
    termios_p->c_cflag &= ~CIBAUDEXT;
    termios_p->c_cflag =
```

```
(termios_p->c_cflag & ~CIBAUD) |
       ((ispeed << IBSHIFT) & CIBAUD);
```

The CSIZE bits specify the character size in bits for both transmission and reception. This size does not include the parity bit, if any. If CSTOPB is set, two stop bits are used; otherwise, one stop bit is used. For example, at 110 baud, two stops bits are required.

If PARENB is set, parity generation and detection is enabled, and a parity bit is added to each character. If parity is enabled, the PARODD flag specifies odd parity if set; otherwise, even parity is used.

If CREAD is set, the receiver is enabled. Otherwise, no characters are received.

If HUPCL is set, the line is disconnected when the last process with the line open closes it or terminates. That is, the data-terminal-ready signal is not asserted.

If CLOCAL is set, the line is assumed to be a local, direct connection with no modem control; otherwise, modem control is assumed.

If CRTSXOFF is set, inbound hardware flow control is enabled.

If CRTSCTS is set, outbound hardware flow control is enabled.

The four possible combinations for the state of CRTSCTS and CRTSXOFF bits and their interactions are described below.

Case A: CRTSCTS off, CRTSXOFF off. In this case the hardware flow control is disabled.

Case B: CRTSCTS on, CRTSXOFF off. In this case only outbound hardware flow control is enabled. The state of CTS signal is used to do outbound flow control. It is expected that output will be suspended if CTS is low and resumed when CTS is high.

Case C: CRTSCTS off, CRTSXOFF on. In this case only inbound hardware flow control is enabled. The state of RTS signal is used to do inbound flow control. It is expected that input will be suspended if RTS is low and resumed when RTS is high.

Case D: CRTSCTS on, CRTSXOFF on. In this case both inbound and outbound hardware flow control are enabled. Uses the state of CTS signal to do outbound flow control and RTS signal to do inbound flow control.

Local Modes

The c_lflag field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline provides the following:

ISIG Enable signals.

ICANON Canonical input (erase and kill processing).

XCASE Canonical upper/lower presentation.

ECHO Enable echo.

ECHOE	Echo erase character as BS-SP-BS &.
ECHOK	Echo NL after kill character.
ECHONL	Echo NL .
NOFLSH	Disable flush after interrupt or quit.
TOSTOP	Send SIGTTOU for background output.
ECHOCTL	Echo control characters as *char,* delete as ^?.
ECHOPRT	Echo erase character as character erased.
ECHOKE	BS-SP-BS erase entire line on line kill.
FLUSHO	Output is being flushed.
PENDIN	Retype pending input at next read or input character.
IEXTEN	Enable extended (implementation-defined) functions.

If ISIG is set, each input character is checked against the special control characters INTR, QUIT, SWTCH, SUSP, STATUS, and DSUSP. If an input character matches one of these control characters, the function associated with that character is performed. (Note: If SWTCH is set and the character matches, the character is simply discarded. No other action is taken.) If ISIG is not set, no checking is done. Thus, these special input functions are possible only if ISIG is set.

If ICANON is set, canonical processing is enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL-c, EOF, EOL, and EOL . If ICANON is not set, read requests are satisfied directly from the input queue. A read is not satisfied until at least MIN characters have been received or the timeout value TIME has expired between characters. This allows fast bursts of input to be read efficiently while still allowing single character input. The time value represents tenths of seconds.

If XCASE is set and ICANON is set, an upper case letter is accepted on input if preceded by a backslash (\) character, and is output preceded by a backslash (\) character. In this mode, the following escape sequences are generated on output and accepted on input:

FOR:	USE:
‘	\'
|	\!
≈	\^
{	\(
}	\)

FOR:	USE:
\	\\

For example, input A as \a, \n as \\n, and \N as \\\n.

If ECHO is set, characters are echoed as received.

When ICANON is set, the following echo functions are possible.

- If ECHO and ECHOE are set, and ECHOPRT is not set, the ERASE and WERASE characters are echoed as one or more ASCII BS SP BS, which clears the last character(s) from a CRT screen.
- If ECHO, ECHOPRT, and IEXTEN are set, the first ERASE and WERASE character in a sequence echoes as a backslash (\), followed by the characters being erased. Subsequent ERASE and WERASE characters echo the characters being erased, in reverse order. The next non-erase character causes a '/' (slash) to be typed before it is echoed. ECHOPRT should be used for hard copy terminals.
- If ECHOKE and IEXTEN are set, the kill character is echoed by erasing each character on the line from the screen (using the mechanism selected by ECHOE and ECHOPRa).
- If ECHOK is set, and ECHOKE is not set, the NL character is echoed after the kill character to emphasize that the line is deleted. Note that a '\' (escape) character or an LNEXT character preceding the erase or kill character removes any special function.
- If ECHONL is set, the NL character is echoed even if ECHO is not set. This is useful for terminals set to local echo (so called half-duplex).

If ECHOCTL and IEXTEN are set, all control characters (characters with codes between 0 and 37 octal) other than ASCII TAB, ASCII NL, the START character, and the STOP character, ASCII CR, and ASCII BS are echoed as ^X, where X is the character given by adding 100 octal to the code of the control character (so that the character with octal code 1 is echoed as ^A), and the ASCII DEL character, with code 177 octal, is echoed as ^?.

If NOFLSH is set, the normal flush of the input and output queues associated with the INTR, QUIT, and SUSP characters is not done. This bit should be set when restarting system calls that read from or write to a terminal (see sigaction(2)).

If TOSTOP and IEXTEN are set, the signal SIGTTOU is sent to a process that tries to write to its controlling terminal if it is not in the foreground process group for that terminal. This signal normally stops the process. Otherwise, the output generated by that process is output to the current output stream. Processes that are blocking or ignoring SIGTTOU signals are excepted and allowed to produce output, if any.

If FLUSHO and IEXTEN are set, data written to the terminal is discarded. This bit is set when the FLUSH character is typed. A program can cancel the effect of typing the FLUSH character by clearing FLUSHO.

If PENDIN and IEXTEN are set, any input that has not yet been read is reprinted when the next character arrives as input. PENDIN is then automatically cleared.

If IEXTEN is set, the following implementation-defined functions are enabled: special characters (WERASE, REPRINT, DISCARD, and LNEXT) and local flags (TOSTOP, ECHOCTL, ECHOPRT, ECHOKE, FLUSHO, and PENDIN).

Minimum and Timeout

The MIN and TIME values were described previously, in the subsection, Non-canonical Mode Input Processing. The initial value of MIN is 1, and the initial value of TIME is 0.

Terminal Size

The number of lines and columns on the terminal's display is specified in the winsize structure defined by sys/termios.h and includes the following members:

```
unsigned    short ws_row;  /* rows, in characters */
unsigned short    ws_col;    /* columns, in characters */
unsigned short    ws_xpixel; /* horizontal size, in pixels */
unsigned short    ws_ypixel; /* vertical size, in pixels */
```

Termio Structure

The SunOS/SVR4 termio structure is used by some ioctls; it is defined by sys/termio.h and includes the following members:

```
unsigned     short    c_iflag;  /* input modes */
unsigned     short    c_oflag;  /* output modes */
unsigned     short    c_cflag;  /* control modes */
unsigned     short c_lflag; /* local modes */
char c_line;              /* line discipline */
unsigned     char  c_cc[NCC];  /* control chars */
```

The special control characters are defined by the array c_cc. The symbolic name NCC is the size of the Control-character array and is also defined by termio.h. The relative positions, subscript names, and typical default values for each function are as follows:

Relative Positions	Subscript Names	Typical Default Values
0	VINTR	EXT
1	VQUIT	FS
2	VERASE	DEL
3	VKILL	NAK
4	VEOF	EOT
5	VEOL	NUL
6	VEOL2	NUL
7	Reserved	

The MIN values is stored in the VMIN element of the c_cc array; the TIME value is stored in the VTIME element of the c_cc array. The VMIN element is the same element as the VEOF element; the VTIME element is the same element as the VEOL element.

The calls that use the termio structure only affect the flags and control characters that can be stored in the termio structure; all other flags and control characters are unaffected.

Modem Lines

On special files representing serial ports, modem control lines can be read. Control lines (if the underlying hardware supports it) may also be changed. Status lines are read-only. The following modem control and status lines may be supported by a device; they are defined by sys/termios.h:

TIOCM_LE	line enable
TIOCM_DTR	data terminal ready
TIOCM_RTS	request to send
TIOCM_ST	secondary transmit
TIOCM_SR	secondary receive
TIOCM_CTS	clear to send
TIOCM_CAR	carrier detect
TIOCM_RNG	ring
TIOCM_DSR	data set ready

TIOCM_CD is a synonym for TIOCM_CAR, and TIOCM_RI is a synonym for TIOCM_RNG. Not all of these are necessarily supported by any particular device; check the manual page for the device in question.

The software carrier mode can be enabled or disabled using the TIOCSSOFTCAR ioctl. If the software carrier flag for a line is off, the line pays attention to the hardware carrier detect (DCD) signal. The tty device associated with the line cannot be opened until DCD is asserted. If the software carrier flag is on, the line behaves as if DCD is always asserted.

The software carrier flag is usually turned on for locally connected terminals or other devices, and is off for lines with modems.

To be able to issue the TIOCGSOFTCAR and TIOCSSOFTCAR ioctl calls, the tty line should be opened with O_NDELAY so that the open(2) will not wait for the carrier.

Default Values

The initial termios values upon driver open is configurable. This is accomplished by setting the "ttymodes" property in the file /kernel/drv/options.conf. Since this property is assigned during system initialization, any change to the "ttymodes" property will not take effect until the next reboot. The string value assigned to this property should be in the same format as the output of the stty(1) command with the -g option.

If this property is undefined, the following `termios` modes are in effect. The initial input control value is `BRKINT`, `ICRNL`, `IXON`, `IMAXBEL`. The initial output control value is `OPOST`, `ONLCR`, `TAB3`. The initial hardware control value is `B9600`, `CS8`, `CREAD`. The initial line-discipline control value is `ISIG`, `ICANON`, `IEXTEN`, `ECHO`, `ECHOK`, `ECHOE`, `ECHOKE`, `ECHOCTL`.

ioctls The `ioctl`s supported by devices and STREAMS modules providing the `termios(3C)` interface are listed below. Some calls may not be supported by all devices or modules. The functionality provided by these calls is also available through the preferred function call interface specified on `termios`.

`TCGETS` The argument is a pointer to a `termios` structure. The current terminal parameters are fetched and stored into that structure.

`TCSETS` The argument is a pointer to a `termios` structure. The current terminal parameters are set from the values stored in that structure. The change is immediate.

`TCSETSW` The argument is a pointer to a `termios` structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted. This form should be used when changing parameters that affect output.

`TCSETSF` The argument is a pointer to a `termios` structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted; all characters queued for input are discarded and then the change occurs.

`TCGETA` The argument is a pointer to a `termio` structure. The current terminal parameters are fetched, and those parameters that can be stored in a `termio` structure are stored into that structure.

`TCSETA` The argument is a pointer to a `termio` structure. Those terminal parameters that can be stored in a `termio` structure are set from the values stored in that structure. The change is immediate.

`TCSETAW` The argument is a pointer to a `termio` structure. Those terminal parameters that can be stored in a `termio` structure are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted. This form should be used when changing parameters that affect output.

`TCSETAF` The argument is a pointer to a `termio` structure. Those terminal parameters that can be stored in a `termio` structure are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted; all characters queued for input are discarded and then the change occurs.

TCSBRK	The argument is an int value. Wait for the output to drain. If the argument is 0, then send a break (zero valued bits for 0.25 seconds).
TCXONC	Start/stop control. The argument is an int value. If the argument is 0, suspend output; if 1, restart suspended output; if 2, suspend input; if 3, restart suspended input.
TCFLSH	The argument is an int value. If the argument is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.
TIOCGPGRP	The argument is a pointer to a pid_t. Set the value of that pid_t to the process group ID of the foreground process group associated with the terminal. See termios(3C) for a description of TCGETPGRP.
TIOCSPGRP	The argument is a pointer to a pid_t. Associate the process group whose process group ID is specified by the value of that pid_t with the terminal. The new process group value must be in the range of valid process group ID values. Otherwise, the error EPERM is returned.
TIOCGSID	The argument is a pointer to a pid_t. The session ID of the terminal is fetched and stored in the pid_t.
TIOCGWINSZ	The argument is a pointer to a winsize structure. The terminal driver's notion of the terminal size is stored into that structure.
TIOCSWINSZ	The argument is a pointer to a winsize structure. The terminal driver's notion of the terminal size is set from the values specified in that structure. If the new sizes are different from the old sizes, a SIGWINCH signal is set to the process group of the terminal.
TIOCMBIS	The argument is a pointer to an int whose value is a mask containing modem control lines to be turned on. The control lines whose bits are set in the argument are turned on; no other control lines are affected.
TIOCMBIC	The argument is a pointer to an int whose value is a mask containing modem control lines to be turned off. The control lines whose bits are set in the argument are turned off; no other control lines are affected.
TIOCMGET	The argument is a pointer to an int. The current state of the modem status lines is fetched and stored in the int pointed to by the argument.
TIOCMSET	The argument is a pointer to an int containing a new set of modem control lines. The modem control lines are turned on or off, depending on whether the bit for that mode is set or clear.
TIOCSPPS	The argument is a pointer to an int that determines whether pulse-per-second event handling is to be enabled (non-zero) or disabled (zero). If a one-pulse-per-second reference clock is attached to the serial

line's data carrier detect input, the local system clock will be calibrated to it. A clock with a high error, that is, a deviation of more than 25 microseconds per tick, is ignored.

`TIOCGPPS` The argument is a pointer to an `int`, in which the state of the even handling is returned. The `int` is set to a non-zero value if pulse-per-second (PPS) handling has been enabled. Otherwise, it is set to zero.

`TIOCGSOFTCAR` The argument is a pointer to an `int` whose value is `1` or `0`, depending on whether the software carrier detect is turned on or off.

`TIOCSSOFTCAR` The argument is a pointer to an `int` whose value is `1` or `0`. The value of the integer should be `0` to turn off software carrier, or `1` to turn it on.

`TIOCGPPSEV` The argument is a pointer to a `struct ppsclockev`. This structure contains the following members:

```
struct timeval tv;
uint32_t serial;
```

"tv" is the system clock timestamp when the event (pulse on the DCD pin) occurred. "serial" is the ordinal of the event, which each consecutive event being assigned the next ordinal. The first event registered gets a "serial" value of `1`. The `TIOCGPPSEV` returns the last event registered; multiple calls will persistently return the same event until a new one is registered. In addition to time stamping and saving the event, if it is of one-second period and of consistently high accuracy, the local system clock will automatically calibrate to it.

Files Files in or under `/dev`

See Also `stty(1)`, `fork(2)`, `getpgid(2)`, `getsid(2)`, `ioctl(2)`, `setsid(2)`, `sigaction(2)`, `signal(3C)`, `tcsetpgrp(3C)`, `termios(3C)`, `signal.h(3HEAD)`, `streamio(7I)`

Name termiox – extended general terminal interface

Description The extended general terminal interface supplements the termio(7I) general terminal interface by adding support for asynchronous hardware flow control, isochronous flow control and clock modes, and local implementations of additional asynchronous features. Some systems may not support all of these capabilities because of either hardware or software limitations. Other systems may not permit certain functions to be disabled. In these cases the appropriate bits will be ignored. See <sys/termiox.h> for your system to find out which capabilities are supported.

Hardware Flow Control Modes

Hardware flow control supplements the termio(7I) IXON, IXOFF, and IXANY character flow control. Character flow control occurs when one device controls the data transfer of another device by the insertion of control characters in the data stream between devices. Hardware flow control occurs when one device controls the data transfer of another device using electrical control signals on wires (circuits) of the asynchronous interface. Isochronous hardware flow control occurs when one device controls the data transfer of another device by asserting or removing the transmit clock signals of that device. Character flow control and hardware flow control may be simultaneously set.

In asynchronous, full duplex applications, the use of the Electronic Industries Association's EIA-232-D Request To Send (RTS) and Clear To Send (CTS) circuits is the preferred method of hardware flow control. An interface to other hardware flow control methods is included to provide a standard interface to these existing methods.

The EIA-232-D standard specified only unidirectional hardware flow control - the Data Circuit-terminating Equipment or Data Communications Equipment (DCE) indicates to the Data Terminal Equipment (DTE) to stop transmitting data. The termiox interface allows both unidirectional and bidirectional hardware flow control; when bidirectional flow control is enabled, either the DCE or DTE can indicate to each other to stop transmitting data across the interface. Note: It is assumed that the asynchronous port is configured as a DTE. If the connected device is also a DTE and not a DCE, then DTE to DTE (for example, terminal or printer connected to computer) hardware flow control is possible by using a null modem to interconnect the appropriate data and control circuits.

Clock Modes

Isochronous communication is a variation of asynchronous communication whereby two communicating devices may provide transmit and/or receive clock signals to one another. Incoming clock signals can be taken from the baud rate generator on the local isochronous port controller, from CCITT V.24 circuit 114, Transmitter Signal Element Timing - DCE source (EIA-232-D pin 15), or from CCITT V.24 circuit 115, Receiver Signal Element Timing - DCE source (EIA-232-D pin 17). Outgoing clock signals can be sent on CCITT V.24 circuit 113, Transmitter Signal Element Timing - DTE source (EIA-232-D pin 24), on CCITT V.24 circuit 128, Receiver Signal Element Timing - DTE source (no EIA-232-D pin), or not sent at all.

In terms of clock modes, traditional asynchronous communication is implemented simply by using the local baud rate generator as the incoming transmit and receive clock source and not outputting any clock signals.

Terminal Parameters

The parameters that control the behavior of devices providing the `termiox` interface are specified by the `termiox` structure defined in the <sys/termiox.h> header. Several `ioctl(2)` system calls that fetch or change these parameters use this structure:

```
#define    NFF    5
struct termiox    {
    unsigned short    x_hflag;       /* hardware flow control modes */
    unsigned short    x_cflag;       /* clock modes */
    unsigned short    x_rflag[NFF];  /* reserved modes */
    unsigned short    x_sflag;       /* spare local modes */
};
```

The `x_hflag` field describes hardware flow control modes:

RTSXOFF	0000001	Enable RTS hardware flow control on input.
CTSXON	0000002	Enable CTS hardware flow control on output.
DTRXOFF	0000004	Enable DTR hardware flow control on input.
CDXON	0000010	Enable CD hardware flow control on output.
ISXOFF	0000020	Enable isochronous hardware flow control on input

The EIA-232-D DTR and CD circuits are used to establish a connection between two systems. The RTS circuit is also used to establish a connection with a modem. Thus, both DTR and RTS are activated when an asynchronous port is opened. If DTR is used for hardware flow control, then RTS must be used for connectivity. If CD is used for hardware flow control, then CTS must be used for connectivity. Thus, RTS and DTR (or CTS and CD) cannot both be used for hardware flow control at the same time. Other mutual exclusions may apply, such as the simultaneous setting of the `termio(7I)` `HUPCL` and the `termiox` DTRXOFF bits, which use the DTE ready line for different functions.

Variations of different hardware flow control methods may be selected by setting the the appropriate bits. For example, bidirectional RTS/CTS flow control is selected by setting both the `RTSXOFF` and `CTSXON` bits and bidirectional DTR/CTS flow control is selected by setting both the `DTRXOFF` and `CTSXON`. Modem control or unidirectional CTS hardware flow control is selected by setting only the `CTSXON` bit.

As previously mentioned, it is assumed that the local asynchronous port (for example, computer) is configured as a DTE. If the connected device (for example, printer) is also a DTE, it is assumed that the device is connected to the computer's asynchronous port using a null modem that swaps control circuits (typically RTS and CTS). The connected DTE drives RTS

and the null modem swaps RTS and CTS so that the remote RTS is received as CTS by the local DTE. In the case that `CTSXON` is set for hardware flow control, printer's lowering of its RTS would cause CTS seen by the computer to be lowered. Output to the printer is suspended until the printer's raising of its RTS, which would cause CTS seen by the computer to be raised.

If `RTSXOFF` is set, the Request To Send (RTS) circuit (line) will be raised, and if the asynchronous port needs to have its input stopped, it will lower the Request To Send (RTS) line. If the RTS line is lowered, it is assumed that the connected device will stop its output until RTS is raised.

If `CTSXON` is set, output will occur only if the Clear To Send (CTS) circuit (line) is raised by the connected device. If the CTS line is lowered by the connected device, output is suspended until CTS is raised.

If `DTRXOFF` is set, the DTE Ready (DTR) circuit (line) will be raised, and if the asynchronous port needs to have its input stopped, it will lower the DTE Ready (DTR) line. If the DTR line is lowered, it is assumed that the connected device will stop its output until DTR is raised.

If `CDXON` is set, output will occur only if the Received Line Signal Detector (CD) circuit (line) is raised by the connected device. If the CD line is lowered by the connected device, output is suspended until CD is raised.

If `ISXOFF` is set, and if the isochronous port needs to have its input stopped, it will stop the outgoing clock signal. It is assumed that the connected device is using this clock signal to create its output. Transit and receive clock sources are programmed using the `x_cflag` fields. If the port is not programmed for external clock generation, `ISXOFF` is ignored. Output isochronous flow control is supported by appropriate clock source programming using the `x_cflag` field and enabled at the remote connected device.

The `x_cflag` field specifies the system treatment of clock modes.

`XMTCLK`	0000007	Transmit clock source:
`XCIBRG`	0000000	Get transmit clock from internal baud rate generator.
`XCTSET`	0000001	Get transmit clock from transmitter signal element timing (DCE source) lead, CCITT V.24 circuit 114, EIA-232-D pin 15.
`XCRSET`	0000002	Get transmit clock from receiver signal element timing (DCE source) lead, CCITT V.24 circuit 115, EIA-232-D pin 17.
`RCVCLK`	0000070	Receive clock source:
`RCIBRG`	0000000	Get receive clock from internal baud rate generator.

RCTSET	0000010	Get receive clock from transmitter signal element timing (DCE source) lead, CCITT V.24 circuit 114, EIA-232-D pin 15.
RCRSET	0000020	Get receive clock from receiver signal element timing (DCE source) lead, CCITT V.24 circuit 115, EIA-232-D pin 17.
TSETCLK	0000700	Transmitter signal element timing (DTE source) lead, CCITT V.24 circuit 113, EIA-232-D pin 24, clock source:
TSETCOFF	0000000	TSET clock not provided.
TSETCRBRG	0000100	Output receive baud rate generator on circuit 113.
TSETCTBRG	0000200	Output transmit baud rate generator on circuit 113
TSETCTSET	0000300	Output transmitter signal element timing (DCE source) on circuit 113.
TSETCRSET	0000400	Output receiver signal element timing (DCE source) on circuit 113.
RSETCLK	0007000	Receiver signal element timing (DTE source) lead, CCITT V.24 circuit 128, no EIA-232-D pin, clock source:
RSETCOFF	0000000	RSET clock not provided.
RSETCRBRG	0001000	Output receive baud rate generator on circuit 128.
RSETCTBRG	0002000	Output transmit baud rate generator on circuit 128.
RSETCTSET	0003000	Output transmitter signal element timing (DCE source) on circuit 128.
RSETCRSET	0004000	Output receiver signal element timing (DCE) on circuit 128.

If the XMTCLK field has a value of XCIBRG the transmit clock is taken from the hardware internal baud rate generator, as in normal asynchronous transmission. If XMTCLK = XCTSET the transmit clock is taken from the Transmitter Signal Element Timing (DCE source) circuit. If XMTCLK = XCRSET the transmit clock is taken from the Receiver Signal Element Timing (DCE source) circuit.

If the RCVCLK field has a value of RCIBRG the receive clock is taken from the hardware Internal Baud Rate Generator, as in normal asynchronous transmission. If RCVCLK = RCTSET the receive clock is taken from the Transmitter Signal Element Timing (DCE source) circuit. If RCVCLK = RCRSET the receive clock is taken from the Receiver Signal Element Timing (DCE source) circuit.

If the TSETCLK field has a value of TSETCOFF the Transmitter Signal Element Timing (DTE source) circuit is not driven. If TSETCLK = TSETCRBRG the Transmitter Signal Element Timing

(DTE source) circuit is driven by the Receive Baud Rate Generator. If TSETCLK = TSETCTBRG the Transmitter Signal Element Timing (DTE source) circuit is driven by the Transmit Baud Rate Generator. If TSETCLK = TSETCTSET the Transmitter Signal Element Timing (DTE source) circuit is driven by the Transmitter Signal Element Timing (DCE source). If TSETCLK = TSETCRBRG the Transmitter Signal Element Timing (DTE source) circuit is driven by the Receiver Signal Element Timing (DCE source).

If the RSETCLK field has a value of RSETCOFF the Receiver Signal Element Timing (DTE source) circuit is not driven. If RSETCLK = RSETCRBRG the Receiver Signal Element Timing (DTE source) circuit is driven by the Receive Baud Rate Generator. If RSETCLK = RSETCTBRG the Receiver Signal Element Timing (DTE source) circuit is driven by the Transmit Baud Rate Generator. If RSETCLK = RSETCTSET the Receiver Signal Element Timing (DTE source) circuit is driven by the Transmitter Signal Element Timing (DCE source). If RSETCLK = RSETCRBRG the Receiver Signal Element Timing (DTE source) circuit is driven by the Receiver Signal Element Timing (DCE source).

The `x_rflag` is reserved for future interface definitions and should not be used by any implementations. The `x_sflag` may be used by local implementations wishing to customize their terminal interface using the `termiox` ioctl system calls.

ioctls The ioctl(2) system calls have the form:

`ioctl (`*fildes, command, arg*`) struct termiox * `*arg*`;`

The commands using this form are:

`TCGETX` The argument is a pointer to a `termiox` structure. The current terminal parameters are fetched and stored into that structure.

`TCSETX` The argument is a pointer to a `termiox` structure. The current terminal parameters are set from the values stored in that structure. The change is immediate.

`TCSETXW` The argument is a pointer to a `termiox` structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted. This form should be used when changing parameters that will affect output.

`TCSETXF` The argument is a pointer to a `termiox` structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted; all characters queued for input are discarded and then the change occurs.

Files `/dev/*`

See Also stty(1), ioctl(2), termio(7I)

Notes The `termiox(7I)` system call is provided for compatibility with previous releases and its use is discouraged. Instead, the `termio(7I)` system call is recommended. See `termio(7I)` for usage information.

Name ticlts, ticots, ticotsord – loopback transport providers

Synopsis

```
#include <sys/ticlts.h>

#include <sys/ticots.h>

#include <sys/ticotsord.h>
```

Description The devices known as `ticlts`, `ticots`, and `ticotsord` are "loopback transport providers," that is, stand-alone networks at the transport level. Loopback transport providers are transport providers in every sense except one: only one host (the local machine) is "connected to" a loopback network. Loopback transports present a TPI (STREAMS-level) interface to application processes and are intended to be accessed via the TLI (application-level) interface. They are implemented as clone devices and support address spaces consisting of "flex-addresses," that is, arbitrary sequences of octets of length > 0, represented by a `netbuf` structure.

`ticlts` is a datagram-mode transport provider. It offers (connectionless) service of type `T_CLTS`. Its default address size is `TCL_DEFAULTADDRSZ`. `ticlts` prints the following error messages (see `t_rcvuderr(3NSL)`):

TCL_BADADDR	bad address specification
TCL_BADOPT	bad option specification
TCL_NOPEER	bound
TCL_PEERBADSTATE	peer in wrong state

`ticots` is a virtual circuit-mode transport provider. It offers (connection-oriented) service of type `T_COTS`. Its default address size is `TCO_DEFAULTADDRSZ`. `ticots` prints the following disconnect messages (see `t_rcvdis(3NSL)`):

TCO_NOPEER	no listener on destination address
TCO_PEERNOROOMONQ	peer has no room on connect queue
TCO_PEERBADSTATE	peer in wrong state
TCO_PEERINITIATED	peer-initiated disconnect
TCO_PROVIDERINITIATED	provider-initiated disconnect

`ticotsord` is a virtual circuit-mode transport provider, offering service of type `T_COTS_ORD` (connection-oriented service with orderly release). Its default address size is `TCOO_DEFAULTADDRSZ`. `ticotsord` prints the following disconnect messages (see `t_rcvdis(3NSL)`):

TCOO_NOPEER	no listener on destination address
TCOO_PEERNOROOMONQ	peer has no room on connect queue

TCOO_PEERBADSTATE	peer in wrong state
TCOO_PEERINITIATED	provider-initiated disconnect
TCOO_PROVIDERINITIATED	peer-initiated disconnect

Usage Loopback transports support a local IPC mechanism through the TLI interface. Applications implemented in a transport provider-independent manner on a client-server model using this IPC are transparently transportable to networked environments.

Transport provider-independent applications must not include the headers listed in the synopsis section above. In particular, the options are (like all transport provider options) provider dependent.

`ticlts` and `ticots` support the same service types (`T_CLTS` and `T_COTS`) supported by the OSI transport-level model.

`ticotsord` supports the same service type (`T_COTSORD`) supported by the TCP/IP model.

Files `/dev/ticlts`

`/dev/ticots`

`/dev/ticotsord`

See Also `t_rcvdis(3NSL)`, `t_rcvuderr(3NSL)`

Name timod – Transport Interface cooperating STREAMS module

Synopsis

```
#include <sys/stropts.h>

ioctl(fildes, I_STR, &my_strioctl);
```

Description `timod` is a STREAMS module for use with the Transport Interface ("TI") functions of the Network Services library. The `timod` module converts a set of `ioctl(2)` calls into STREAMS messages that may be consumed by a transport protocol provider that supports the Transport Interface. This allows a user to initiate certain TI functions as atomic operations.

The `timod` module must be pushed onto only a stream terminated by a transport protocol provider that supports the TI.

All STREAMS messages, with the exception of the message types generated from the `ioctl` commands described below, will be transparently passed to the neighboring module or driver. The messages generated from the following `ioctl` commands are recognized and processed by the `timod` module. The format of the `ioctl` call is:

```
#include <sys/stropts.h>
        -
        -
struct strioctl my_strioctl;
        -
        -
strioctl.ic_cmd = cmd;
strioctl.ic_timout = INFTIM;
strioctl.ic_len = size;
strioctl.ic_dp = (char *)buf
ioctl(fildes, I_STR, &my_strioctl);
```

On issuance, `size` is the size of the appropriate TI message to be sent to the transport provider and on return `size` is the size of the appropriate TI message from the transport provider in response to the issued TI message. *buf* is a pointer to a buffer large enough to hold the contents of the appropriate TI messages. The TI message types are defined in `<sys/tihdr.h>`. The possible values for the *cmd* field are:

`TI_BIND` Bind an address to the underlying transport protocol provider. The message issued to the `TI_BIND` ioctl is equivalent to the TI message type `T_BIND_REQ` and the message returned by the successful completion of the `ioctl` is equivalent to the TI message type `T_BIND_ACK`.

`TI_UNBIND` Unbind an address from the underlying transport protocol provider. The message issued to the `TI_UNBIND` ioctl is equivalent to the TI message type `T_UNBIND_REQ` and the message returned by the successful completion of the `ioctl` is equivalent to the TI message type `T_OK_ACK`.

`TI_GETINFO` Get the TI protocol specific information from the transport protocol provider. The message issued to the `TI_GETINFO` ioctl is equivalent to the TI

message type T_INFO_REQ and the message returned by the successful completion of the ioctl is equivalent to the TI message type T_INFO_ACK.

TI_OPTMGMT — Get, set, or negotiate protocol specific options with the transport protocol provider. The message issued to the TI_OPTMGMT ioctl is equivalent to the TI message type T_OPTMGMT_REQ and the message returned by the successful completion of the ioctl is equivalent to the TI message type T_OPTMGMT_ACK.

Files

<sys/timod.h>	ioctl definitions
<sys/tiuser.h>	TLI interface declaration and structure file
<sys/tihdr.h>	TPI declarations and user-level code
<sys/errno.h>	system error messages file. Please see errno(3C).

See Also Intro(3), ioctl(2), errno(3C), tirdwr(7M)

STREAMS Programming Guide

Diagnostics If the ioctl returns with a value greater than 0, the lower 8 bits of the return value will be one of the TI error codes as defined in <sys/tiuser.h>. If the TI error is of type TSYSERR, then the next 8 bits of the return value will contain an error as defined in <sys/errno.h> (see Intro(3)).

Name tirdwr – Transport Interface read/write interface STREAMS module

Synopsis

```
int ioctl( fd, I_PUSH, "tirdwr");
```

Description `tirdwr` is a STREAMS module that provides an alternate interface to a transport provider which supports the Transport Interface ("TI") functions of the Network Services library (see Section 3N). This alternate interface allows a user to communicate with the transport protocol provider using the `read(2)` and `write(2)` system calls. The `putmsg(2)` and `getmsg(2)` system calls may also be used. However, `putmsg` and `getmsg` can only transfer data messages between user and stream; control portions are disallowed.

The `tirdwr` module must only be pushed (see `I_PUSH` in `streamio(7I)`) onto a stream terminated by a transport protocol provider which supports the TI. After the `tirdwr` module has been pushed onto a stream, none of the TI functions can be used. Subsequent calls to TI functions cause an error on the stream. Once the error is detected, subsequent system calls on the stream return an error with `errno` set to `EPROTO`.

The following are the actions taken by the `tirdwr` module when pushed on the stream, popped (see `I_POP` in `streamio(7I)`) off the stream, or when data passes through it.

`push` When the module is pushed onto a stream, it checks any existing data destined for the user to ensure that only regular data messages are present. It ignores any messages on the stream that relate to process management, such as messages that generate signals to the user processes associated with the stream. If any other messages are present, the `I_PUSH` will return an error with `errno` set to `EPROTO`.

`write` The module takes the following actions on data that originated from a `write` system call:

- All messages with the exception of messages that contain control portions (see the `putmsg` and `getmsg` system calls) are transparently passed onto the module's downstream neighbor.
- Any zero length data messages are freed by the module and they will not be passed onto the module's downstream neighbor.
- Any messages with control portions generate an error, and any further system calls associated with the stream fails with `errno` set to `EPROTO`.

`read` The module takes the following actions on data that originated from the transport protocol provider.

All messages with the exception of those that contain control portions (see the `putmsg` and `getmsg` system calls) are transparently passed onto the module's upstream neighbor. The action taken on messages with control portions will be as follows:

- Any data messages with control portions have the control portions removed from the message before to passing the message on to the upstream neighbor.

- Messages that represent an orderly release indication from the transport provider generate a zero length data message, indicating the end of file, which will be sent to the reader of the stream. The orderly release message itself is freed by the module.
- Messages that represent an abortive disconnect indication from the transport provider cause all further `write` and `putmsg` system calls to fail with `errno` set to `ENXIO`. All further `read` and `getmsg` system calls return zero length data (indicating end of file) once all previous data has been read.
- With the exception of the above rules, all other messages with control portions generate an error and all further system calls associated with the stream will fail with `errno` set to `EPROTO`.

Any zero length data messages are freed by the module and they are not passed onto the module's upstream neighbor.

`pop` When the module is popped off the stream or the stream is closed, the module takes the following action:

- If an orderly release indication has been previously received, then an orderly release request will be sent to the remote side of the transport connection.

See Also `Intro(3)`, `getmsg(2)`, `putmsg(2)`, `read(2)`, `write(2)`, `Intro(3)`, `streamio(7I)`, `timod(7M)`

STREAMS Programming Guide

Name tmpfs – memory based file system

Synopsis

```
#include <sys/mount.h>

mount (special, directory, MS_DATA, "tmpfs", NULL, 0);
```

Description `tmpfs` is a memory based file system which uses kernel resources relating to the VM system and page cache as a file system. Once mounted, a `tmpfs` file system provides standard file operations and semantics. `tmpfs` is so named because files and directories are not preserved across reboot or unmounts, all files residing on a `tmpfs` file system that is unmounted will be lost.

`tmpfs` file systems can be mounted with the command:

```
mount -F tmpfs swap directory
```

Alternatively, to mount a `tmpfs` file system on `/tmp` at multi-user startup time (maximizing possible performance improvements), add the following line to `/etc/vfstab`:

```
swap –/tmp tmpfs – yes –
```

`tmpfs` is designed as a performance enhancement which is achieved by caching the writes to files residing on a `tmpfs` file system. Performance improvements are most noticeable when a large number of short lived files are written and accessed on a `tmpfs` file system. Large compilations with `tmpfs` mounted on `/tmp` are a good example of this.

Users of `tmpfs` should be aware of some constraints involved in mounting a `tmpfs` file system. The resources used by `tmpfs` are the same as those used when commands are executed (for example, swap space allocation). This means that large sized `tmpfs` files can affect the amount of space left over for programs to execute. Likewise, programs requiring large amounts of memory use up the space available to `tmpfs`. Users running into this constraint (for example, running out of space on `tmpfs`) can allocate more swap space by using the swap(1M) command.

Another constraint is that the number of files available in a `tmpfs` file system is calculated based on the physical memory of the machine and not the size of the swap device/partition. If you have too many files, `tmpfs` will print a warning message and you will be unable to create new files. You cannot increase this limit by adding swap space.

Normal file system writes are scheduled to be written to a permanent storage medium along with all control information associated with the file (for example, modification time, file permissions). `tmpfs` control information resides only in memory and never needs to be written to permanent storage. File data remains in core until memory demands are sufficient to cause pages associated with `tmpfs` to be reused at which time they are copied out to swap.

An additional mount option can be specified to control the size of an individual `tmpfs` file system.

See Also df(1M), mount(1M), mount_tmpfs(1M), swap(1M), mmap(2), mount(2), umount(2), vfstab(4)

System Administration Guide: Basic Administration

Diagnostics If `tmpfs` runs out of space, one of the following messages will display in the console.

directory`: File system full, swap space limit exceeded`
This message appears because a page could not be allocated while writing to a file. This can occur if `tmpfs` is attempting to write more than it is allowed, or if currently executing programs are using a lot of memory. To make more space available, remove unnecessary files, exit from some programs, or allocate more swap space using swap(1M).

directory`: File system full, memory allocation failed`
`tmpfs` ran out of physical memory while attempting to create a new file or directory. Remove unnecessary files or directories or install more physical memory.

Warnings Files and directories on a `tmpfs` file system are not preserved across reboots or unmounts. Command scripts or programs which count on this will not work as expected.

Notes Compilers do not necessarily use `/tmp` to write intermediate files therefore missing some significant performance benefits. This can be remedied by setting the environment variable `TMPDIR` to `/tmp`. Compilers use the value in this environment variable as the name of the directory to store intermediate files.

`swap` to a `tmpfs` file is not supported.

df(1M) output is of limited accuracy since a `tmpfs` file system size is not static and the space available to `tmpfs` is dependent on the swap space demands of the entire system.

Name todopol – Time-Of-Day driver for SPARC Enterprise Server family

Description The `todopl` driver is the Time-Of-Day (TOD) driver for the SPARC Enterprise Server family.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcakr
Interface Stability	Private

See Also `attributes(5)`

Name tokenmt – Single and Two Rate Three Conformance Level Meter

Description The `tokenmt` module can be configured as a Single or a Two Rate meter. Packets are deemed to belong to one of the three levels - Red, Yellow or Green - depending on the configured rate(s) and the burst sizes. When configured as a Single Rate meter, `tokenmt` can operate with just the Green and Red levels.

Configuration parameters for `tokenmt` correspond to definitions in *RFC– 2697* and *RFC– 2698* as follows:

Configuring `tokenmt` as a Single Rate meter (from *RFC– 2697*):

committed_rate - CIR
committed_burst - CBS
peak_burst - EBS

(thus 'peak_burst' for a single rate meter is actually the 'excess burst' in the RFC. However, throughout the text the parameter name “peak burst” is used.)

Configuring tokenmt as a Two Rate meter (from *RFC– 2698*):

committed_rate - CIR
peak_rate - PIR
committed_burst - CBS
peak_burst - PBS

The meter is implemented using token buckets C and P, which initially hold tokens equivalent to committed and peak burst sizes (bits) respectively. When a packet of size *B* bits arrive at time *t*, the following occurs:

```
When operating as a Single Rate meter, the outcome (level)
is decided as follows:
        - Update tokens in C and P
              o Compute no. of tokens accumulated since the
                last time packet was seen at the committed rate as
                T(t) = committed rate * (t - t')
                (where t' is the time the last packet was seen)
              o Add T tokens to C up to a maximum of committed burst
                size. Add remaining tokens ((C+T) - Commited Burst),
                if any, to P, to a maximum of peak burst size.
        - Decide outcome
              o If not color aware
                      o If B <= C, outcome is GREEN and C -= B.
                      o Else, if B <= P, outcome is YELLOW and P -= B.
                      o Else, outcome is Red.
              o Else,
                      o obtain DSCP from packet
```

```
                    o obtain color from color_map, color_map[DSCP]
                    o if (color is GREEN) and (B <= C), outcome is
                      GREEN and C -= B.
                    o Else, if (color is GREEN or YELLOW) and
                      (B <= P), outcome is YELLOW and  P -= B.
                    o Else, outcome is RED.
    Note that if peak_burst and yellow_next_actions are
    not specified (that is, a single rate meter with two
    outcomes), the outcome is never YELLOW.
```

When operating as a Two Rate meter, the outcome (level) is decided as follows:

```
- Update tokens in C and P
                o Compute no. of tokens accumulated since the last time a
                  packet was seen at the committed and peak rates as
                  Tc(t) = committed rate * (t - t')
                  Tp(t) = peak rate * (t - t')
                  (where t' is the time the last packet was seen)
                o Add Tc to C up to a maximum of committed burst size
                o Add Tp to P up to a maximum of peak burst size
            - Decide outcome
                o If not color aware
                      o If B > P, outcome is RED.
                      o Else, if B > C, outcome is YELLOW and P -= B
                      o Else, outcome is GREEN and C -= B & P -= B
                o Else,
                      o obtain DSCP from packet
                      o obtain color from color_map, color_map[DSCP]
                      o if (color is RED) or (B > P), outcome is RED
                      o Else, if (color is YELLOW) or (B > C),
                        outcome is YELLOW and P -= B
                      o Else, outcome is GREEN and C -= B & P -= B
```

Statistics The `tokenmt` module exports the following statistics through `kstat`:

Global statistics:

```
module: tokenmt                         instance: <action id>
  name: tokenmt statistics              class <action name>
        epackets                        <number of packets in error>
        green_bits                      <number of bits in green>
        green_packets                   <number of packets in green>
        red_bits                        <number of bits in red>
        red_packets                     <number of packets in red>
        yellow_bits                     <number of bits in yellow>
        yellow packets                  <number of packets in yellow>
```

Files `/kernel/ipp/sparcv9/tokenmt` 64-bit module (SPARC only.)

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWqos

See Also `ipqosconf(1M)`, `dlcosmk(7ipp)`, `dscpmk(7ipp)`, `flowacct(7ipp)`, `ipqos(7ipp)`, `ipgpc(7ipp)`, `tswtclmt(7ipp)`

RFC 2697, A Single Rate Three Color Marker J. Heinanen, R. Guerin — The Internet Society, 1999

RFC 2698, A Two Rate Three Color Marker J. Heinanen, R. Guerin — The Internet Society, 1999

Name tpf – Platform Specific Module (PSM) for Tricord Systems Enterprise Server Models ES3000, ES4000 and ES5000.

Description `tpf` provides the platform dependent functions for Solaris x86 MP support. These functions adhere to the PSMI Specifications. (Platform Specific Module Interface Specifications.) Tricord Systems Enterprise Servers are Intel APIC based MP platforms which run from 1 to 12 Intel processors. The `tpf` psm supports dynamic interrupt distribution across all processors in an MP configuration.

The psm is automatically invoked on an ESxxxx platform at system boot time.

Files `/kernel/mach/tpf` MP module.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86

See Also `attributes(5)`

Name tsalarm – Alarm device driver

Synopsis `tsalarm@0:ctl`

Description The `tsalarm` driver is a Multi-threaded, loadable non-STREAMS pseudo driver that manages ALOM alarms. The `tsalarm` driver provides an interface through which alarm relays can be controlled on SUNW,Netra-240 and SUNW,Netra-440 platforms.

Hardware Interface The alarm hardware differs depending on platform. The Netra 240 and 440 platforms features four dry contact alarm relays which are controlled by ALOM. You can set each alarm to "on" or "off" by using ioctl interfaces provided from the host. The four alarms are labeled as "critical," "major," "minor," and "user." The user alarm is set by a user application depending on system condition. LED's in front of the box provide a visual indication of the four alarms. The number of alarms and their meanings/labels may vary across platforms.

ioctls The interface provided by the `tsalarm` driver comprises ioctls that enable applications to manipulate the alarm module. The alarm module is accessed via two device nodes: i) `/dev/lom` and `/dev/tsalarm:ctl`.

The following ioctls are supported by the `/dev/lom` and `/dev/tsalarm:ctl` devices:

`TSIOCALCTL - Turn an alarm on or off.`

The argument is a pointer to the `ts_aldata_t/lom_aldata_t` structure. This structure is described below. `alarm_no member` is an integer which specifies the alarm to which the command is to be applied. The `alarm_state/state` structure member indicates the state to which the alarm should be set (where 0 == off). An error (`EINVAL`) is returned if either an invalid alarm_no or invalid alarm_state is provided.

`TSIOCALSTATE - Get the state of the alarms.`

The argument is a pointer to the `ts_aldata_t/lom_aldata_t` structure. This structure is described below. `alarm_no member` is an integer which indicates the alarm to which the command will be applied. The `alarm_state` member holds the alarm's current state and is filled in by the driver. A zero indicates that the alarm is off. An error (`EINVAL`) is returned if an invalid alarm_no is provided. The structures and definitions for the values are defined

below.

Alarm values:

```
The following old style values are defined in <lom.io.h>

     #define ALARM_NUM_0         0 /* number of zero'th alarm */

     #define ALARM_NUM_1         1 /* number of first alarm */

     #define ALARM_NUM_2         2 /* number of second alarm */

     #define ALARM_NUM_3         3 /* number of third alarm */

  Alarm values defined in <lom.io.h>

     #define ALARM_OFF          0 /* Turn off alarm */

     #define ALARM_ON           1 /* Turn on alarm */
```

Alarm Data Structure:

```
This structure is defined in <lom.io.h>

     typedef struct {

          int alarm_no;       /* alarm to apply command to */

          int alarm_state;    /* state of alarm (0 == off) */

     } ts_aldata_t;
```

Use the following LOM interfaces to get and set the alarms. These definitions are included in <lom_io.h>

```
    #define ALARM_CRITICAL         0 /* number of critical alarm */

    #define ALARM_MAJOR            1 /* number of major alarm */

    #define ALARM_MINOR            2 /* number of minor alarm */

    #define ALARM_USER             3 /* number of user alarm */
```

The following alarm data structure is provided in <lom_io.h>:

```
typedef struct {

        int alarm_no;
```

```
        int state;

    } lom_aldata_t;
```

Errors An `open()` will fail if:

ENXIO The driver is not installed in the system.

An `ioctl()` will fail if:

EFAULT There was a hardware failure during the specified operation.

EINVAL The alarm number specified is not valid or an invalid value was supplied.

ENXIO The driver is not installed in the system or the monitor callback routine could not be scheduled.

Examples How to set an alarm:

```
#include <sys/unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <lom_io.h>

#define LOM_DEVICE "/dev/lom"

int
main()
{
   lom_aldata_t lld;
   int fd = open(LOM_DEVICE, O_RDWR);

   if (fd == -1) {
           printf("Error opening device: %s\n", LOM_DEVICE);
           exit (1);
   }

   lld.alarm_no = ALARM_CRITICAL;   /* Set the critical alarm */
   lld.state = ALARM_ON;   /* Set the alarm */

   if (ioctl(fd, LOMIOCALCTL, (char *)&lld) != 0)
           printf("Setting alarm failed");
   else
           printf("Alarm set successfully");

   close(fd);

}
```

Files

`/dev/lom`
LOM device.

`/dev/tsalarm:ctl`
Alarm control device.

`/platform/platform/kernel/drv/sparcv9/tsalarm`
Device driver module.

`/platform/SUNW,Netra-240/kernel/drv/tsalarm.conf`
Driver configuration file.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcarx.u
Architecture	SPARC

See Also `attributes(5)`

Writing Device Drivers

Name tswtclmt – Time Sliding Window Three Conformance Level Meter

Description The Time Sliding Window Three Conformance level meter (`tswtcl`) meters a traffic stream and determines the conformance level of its packets.

Packets are deemed to belong to one of the three levels, Red, Yellow or Green, depending on the committed and peak rate.

The meter provides an estimate of the running average bandwidth. It takes into account burstiness and smoothes out its estimate to approximate the longer-term measured sending rate of the traffic stream.

The estimated bandwidth approximates the running average bandwidth of the traffic stream over a specific window (time interval). `tswtcl` estimates the average bandwidth using a time-based estimator. When a packet arrives for a class, `tswtcl` re-computes the average rate by using the rate in the last window (time interval) and the size of the arriving packet. The window is then slid to start at the current time (the packet arrival time). If the computed rate is less than the committed configuration parameter, the packet is deemed Green; else if the rate is less than the peak rate, it is Yellow; else Red. To avoid dropping multiple packets within a TCP window, `tswtcl` probabilistically assigns one of the three conformance level to the packet.

Statistics The `tswtcl` module exports global and per-class statistics through kstat:

Global statistics:

```
module: tswtclmt                        instance: <action id>
  name: tswtclmt statistics             class <action name>
        green_bits                      <number of bit in green>
        green_packets                   <number of packets in green>
        red_bits                        <number of bits in red>
        red_packets                     <number of packets in red>
        yellow_bits                     <number of bits in yellow>
        yellow_packets                  <number of packets in yellow>
```

Files `/kernel/ipp/sparcv9/tswtclmt` 64-bit module (SPARC only.)

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWqos

See Also `ipqosconf(1M)`, `dlcosmk(7ipp)`, `dscpmk(7ipp)`, `flowacct(7ipp)`, `ipqos(7ipp)`, `ipgpc(7ipp)`, `tokenmt(7ipp)`

RFC 2859, A Time Sliding Window Three Colour Marker (TSWTCM) W. Fang, N. Seddigh, B. Nandy — The Internet Society, 2000

Name ttcompat – V7, 4BSD and XENIX STREAMS compatibility module

Synopsis

```
#define BSD_COMP
#include <sys/stropts.h>
#include <sys/ioctl.h>
ioctl(fd, I_PUSH, "ttcompat");
```

Description `ttcompat` is a STREAMS module that translates the `ioctl` calls supported by the older `Version 7, 4BSD,` and `XENIX` terminal drivers into the `ioctl` calls supported by the `termio` interface (see `termio(7I)`). All other messages pass through this module unchanged; the behavior of `read` and `write` calls is unchanged, as is the behavior of `ioctl` calls other than the ones supported by `ttcompat`.

This module can be automatically pushed onto a stream using the `autopush` mechanism when a terminal device is opened; it does not have to be explicitly pushed onto a stream. This module requires that the `termios` interface be supported by the modules and the application can push the driver downstream. The `TCGETS, TCSETS,` and `TCSETSF` `ioctl` calls must be supported. If any information set or fetched by those `ioctl` calls is not supported by the modules and driver downstream, some of the `V7/4BSD/XENIX` functions may not be supported. For example, if the `CBAUD` bits in the `c_cflag` field are not supported, the functions provided by the `sg_ispeed` and `sg_ospeed` fields of the `sgttyb` structure (see below) will not be supported. If the `TCFLSH` `ioctl` is not supported, the function provided by the `TIOCFLUSH` `ioctl` will not be supported. If the `TCXONC` `ioctl` is not supported, the functions provided by the `TIOCSTOP` and `TIOCSTART` `ioctl` calls will not be supported. If the `TIOCMBIS` and `TIOCMBIC` `ioctl` calls are not supported, the functions provided by the `TIOCSDTR` and `TIOCCDTR` `ioctl` calls will not be supported.

The basic `ioctl` calls use the `sgttyb` structure defined by <`sys/ttold.h`> (included by <`sys/ioctl.h`>):

```
struct   sgttyb   {
    char    sg_ispeed;
    char    sg_ospeed;
    char    sg_erase;
    char    sg_kill;
    int    sg_flags;
};
```

The `sg_ispeed` and `sg_ospeed` fields describe the input and output speeds of the device. If the speed set on the device is over B38400, then it is reported as B38400 for compatibility reasons. If it is set to B38400 and the current speed is over B38400, the change is ignored. See TIOCGETP and TIOCSETP below. The `sg_erase` and `sg_kill` `fields` of the argument structure specify the erase and kill characters respectively, and reflect the values in the VERASE and VKILL members of the `c_cc` `field` of the `termios` structure.

The `sg_flags` field of the argument structure contains several flags that determine the system's treatment of the terminal. They are mapped into flags in fields of the terminal state, represented by the `termios` structure.

Delay type `0` (NL0, TAB0, CR0, FF0, BS0) is always mapped into the equivalent delay type `0` in the `c_oflag` field of the `termios` structure. Other delay mappings are performed as follows:

sg_flags	c_oflag
BS1	BS1
FF1	VT1
CR1	CR2
CR2	CR3
CR3	CR0 (not supported)
TAB1	TAB1
TAB2	TAB2
XTABS	TAB3
NL1	ONLRET|CR1
NL2	NL1
NL3	NL0 (not supported)

If previous `TIOCLSET` or `TIOCLBIS` `ioctl` calls have not selected `LITOUT` or `PASS8` mode, and if `RAW` mode is not selected, the `ISTRIP` flag is set in the `c_iflag` field of the `termios` structure, and the `EVENP` and `ODDP` flags control the parity of characters sent to the terminal and accepted from the terminal, as follows:

0 (neither EVENP nor ODDP)
: Parity is not to be generated on output or checked on input. The character size is set to `CS8` and the `PARENB` flag is cleared in the `c_cflag` field of the `termios` structure.

EVENP
: Even parity characters are to be generated on output and accepted on input. The INPCK flag is set in the `c_iflag` field of the `termios` structure, the character size is set to `CS7` and the `PARENB` flag is set in the `c_iflag` field of the `termios` structure.

ODDP
: Odd parity characters are to be generated on output and accepted on input. The `INPCK` flag is set in the `c_iflag`, the character size is set to `CS7` and the `PARENB` and `PARODD` flags are set in the `c_iflag` field of the `termios` structure.

EVENP|ODDP or ANYP
: Even parity characters are to be generated on output and characters of either parity are to be accepted on input. The `INPCK` flag is cleared in the `c_iflag` field, the character size

is set to CS7 and the PARENB flag is set in the c_iflag field of the termios structure.

The RAW flag disables all output processing (the OPOST flag in the c_oflag field, and the XCASE and IEXTEN flags in the c_iflag field are cleared in the termios structure) and input processing (all flags in the c_iflag field other than the IXOFF and IXANY flags are cleared in the termios structure). Eight bits of data, with no parity bit are accepted on input and generated on output; the character size is set to CS8 and the PARENB and PARODD flags are cleared in the c_cflag field of the termios structure. The signal-generating and line-editing control characters are disabled by clearing the ISIG and ICANON flags in the c_iflag field of the termios structure.

The CRMOD flag turns input carriage return characters into linefeed characters, and output linefeed characters to be sent as a carriage return followed by a linefeed. The ICRNL flag in the c_iflag field, and the OPOST and ONLCR flags in the c_oflag field, are set in the termios structure.

The LCASE flag maps upper-case letters in the ASCII character set to their lower-case equivalents on input (the IUCLC flag is set in the c_iflag field), and maps lower-case letters in the ASCII character set to their upper-case equivalents on output (the OLCUC flag is set in the c_oflag field). Escape sequences are accepted on input, and generated on output, to handle certain ASCII characters not supported by older terminals (the XCASE flag is set in the c_lflag field).

Other flags are directly mapped to flags in the termios structure:

sg_flags	Flags in termios structure
CBREAK	Complement of ICANON in c_lflag field
ECHO	ECHO in c_lflag field
TANDEM	IXOFF in c_iflag field

Another structure associated with each terminal specifies characters that are special in both the old Version 7 and the newer 4BSD terminal interfaces. The following structure is defined by <sys/ttold.h>:

```
struct   tchars   {
          char t_intrc;     /* interrupt */
          char t_quitc;     /* quit */
          char t_startc; /* start output */
          char t_stopc;     /* stop output */
          char t_eofc;        /* end-of-file */
          char t_brkc;        /* input delimiter (like nl) */
     };
```

XENIX defines the tchar structure as tc. The characters are mapped to members of the c_cc field of the termios structure as follows:

```
tchars          c_cc index
t_intrc         VINTR
t_quitc         VQUIT
t_startc        VSTART
t_stopc         VSTOP
t_eofc          VEOF
t_brkc          VEOL
```

Also associated with each terminal is a local flag word (TIOCLSET and TIOCLGET), specifying flags supported by the new 4BSD terminal interface. Most of these flags are directly mapped to flags in the termios structure:

Local flags	Flags in termios structure
LCRTBS	Not supported
LPRTERA	ECHOPRT in the c_lflag field
LCRTERA	ECHOE in the c_lflag field
LTILDE	Not supported
LMDMBUF	Not supported
LTOSTOP	TOSTOP in the c_lflag field
LFLUSHO	FLUSHO in the c_lflag field
LNOHANG	CLOCAL in the c_cflag field
LCRTKIL	ECHOKE in the c_lflag field
LPASS8	CS8 in the c_cflag field
LCTLECH	CTLECH in the c_lflag field
LPENDIN	PENDIN in the c_lflag field
LDECCTQ	Complement of IXANY in the c_iflag field
LNOFLSH	NOFLSH in the c_lflag field

Each flag has a corresponding equivalent sg_flags value. The sg_flags definitions omit the leading "L"; for example, TIOCSETP with sg_flags set to TOSTOP is equivalent to TIOCLSET with LTOSTOP.

Another structure associated with each terminal is the ltchars structure which defines control characters for the new 4BSD terminal interface. Its structure is:

```
struct ltchars {
   char t_suspc;  /* stop process signal */
   char t_dsuspc; /* delayed stop process signal */
   char t_rprntc; /* reprint line */
   char t_flushc; /*flush output (toggles) */
   char t_werasc; /* word erase */
   char t_lnextc; /* literal next character */
};
```

The characters are mapped to members of the `c_cc` field of the `termios` structure as follows:

ltchars	c_cc index
t_suspc	VSUS
t_dsuspc	VDSUSP
t_rprntc	VREPRINT
t_flushc	VDISCARD
t_werasc	VWERASE
t_lnextc	VLNEXT

ioctls `ttcompat` responds to the following `ioctl` calls. All others are passed to the module below.

`TIOCGETP` The argument is a pointer to an `sgttyb` structure. The current terminal state is fetched; the appropriate characters in the terminal state are stored in that structure, as are the input and output speeds. If the speed is over B38400, then B38400 is returned. The values of the flags in the `sg_flags` field are derived from the flags in the terminal state and stored in the structure.

`TIOCEXCL` Set "exclusive-use" mode; no further opens are permitted until the file has been closed.

`TIOCNXCL` Turn off "exclusive-use" mode.

`TIOCSETP` The argument is a pointer to an `sgttyb` structure. The appropriate characters and input and output speeds in the terminal state are set from the values in that structure, and the flags in the terminal state are set to match the values of the flags in the `sg_flags` field of that structure. The state is changed with a `TCSETSF` `ioctl` so that the interface delays until output is quiescent, then throws away any unread characters, before changing the modes. If the current device speed is over B38400 for either input or output speed, and B38400 is specified through this interface for that speed, the actual device speed is not changed. If the device speed is B38400 or lower or if some speed other than B38400 is specified, then the actual speed specified is set.

`TIOCSETN` The argument is a pointer to an `sgttyb` structure. The terminal state is changed as `TIOCSETP` would change it, but a `TCSETS` `ioctl` is used, so that the interface neither delays nor discards input.

`TIOCHPCL` The argument is ignored. The `HUPCL` flag is set in the `c_cflag` word of the terminal state.

`TIOCFLUSH` The argument is a pointer to an `int` variable. If its value is zero, all characters waiting in input or output queues are flushed. Otherwise, the value of the `int` is treated as the logical `OR` of the `FREAD` and `FWRITE` flags defined by `<sys/file.h>`. If the `FREAD` bit is set, all characters waiting in input queues are flushed, and if the `FWRITE` bit is set, all characters waiting in output queues are flushed.

`TIOCSBRK` The argument is ignored. The break bit is set for the device. (This is not supported by `ttcompat`. The underlying driver must support TIOCSBRK.)

`TIOCCBRK` The argument is ignored. The break bit is cleared for the device. (This is not supported by `ttcompat`. The underlying driver must support TIOCCBRK.)

`TIOCSDTR` The argument is ignored. The Data Terminal Ready bit is set for the device.

`TIOCCDTR` The argument is ignored. The Data Terminal Ready bit is cleared for the device.

`TIOCSTOP` The argument is ignored. Output is stopped as if the STOP character had been typed.

`TIOCSTART` The argument is ignored. Output is restarted as if the START character had been typed.

`TIOCGETC` The argument is a pointer to a `tchars` structure. The current terminal state is fetched, and the appropriate characters in the terminal state are stored in that structure.

`TIOCSETC` The argument is a pointer to a `tchars` structure. The values of the appropriate characters in the terminal state are set from the characters in that structure.

`TIOCLGET` The argument is a pointer to an `int`. The current terminal state is fetched, and the values of the local flags are derived from the flags in the terminal state and stored in the `int` pointed to by the argument.

`TIOCLBIS` The argument is a pointer to an `int` whose value is a mask containing flags to be set in the local flags word. The current terminal state is fetched, and the values of the local flags are derived from the flags in the terminal state; the specified flags are set, and the flags in the terminal state are set to match the new value of the local flags word.

`TIOCLBIC` The argument is a pointer to an `int` whose value is a mask containing flags to be cleared in the local flags word. The current terminal state is fetched, and the

values of the local flags are derived from the flags in the terminal state; the specified flags are cleared, and the flags in the terminal state are set to match the new value of the local flags word.

`TIOCLSET` The argument is a pointer to an int containing a new set of local flags. The flags in the terminal state are set to match the new value of the local flags word. (This `ioctl` was added because `sg_flags` was once a 16 bit value. The local modes controlled by TIOCLSET are equivalent to the modes controlled by TIOCSETP and `sg_flags`.)

`TIOCGLTC` The argument is a pointer to an `ltchars` structure. The values of the appropriate characters in the terminal state are stored in that structure.

`TIOCSLTC` The argument is a pointer to an `ltchars` structure. The values of the appropriate characters in the terminal state are set from the characters in that structure.

`FIORDCHK` Returns the number of immediately readable characters. The argument is ignored. (This ioctl is handled in the stream head, not in the `ttcompat` module.)

`FIONREAD` Returns the number of immediately readable characters in the int pointed to by the argument. (This ioctl is handled in the stream head, not in the `ttcompat` module.)

The following ioctls are returned as successful for the sake of compatibility. However, nothing significant is done (that is, the state of the terminal is not changed in any way, and no message is passed through to the underlying `tty` driver).

```
DIOCSETP
DIOCSETP
DIOCGETP
LDCLOSE
LDCHG
LDOPEN
LDGETT
LDSETT
TIOCGETD
TIOCSETD
```

The following old `ioctl`s are not supported by `ttcompat`, but are supported by Solaris `tty` drivers. As with all ioctl not otherwise listed in this documentation, these are passed through to the underlying driver and are handled there.

```
TIOCREMOTE
TIOCGWINSZ
TIOCSWINSZ
```

The following `ioctls` are not supported by `ttcompat`, and are generally not supported by Solaris `tty` drivers. They are passed through, and the `tty` drivers return EINVAL.

```
LDSMAP
LDGMAP
LDNMAP
TIOCNOTTY
TIOCOUTQ
```

(Note: LDSMAP, LDGMAP, and LDNMAP are defined in <`sys/termios.h`>.)

See Also `ioctl(2)`, `termios(3C)`, `ldterm(7M)`, `termio(7I)`

Name tty – controlling terminal interface

Description The file `/dev/tty` is, in each process, a synonym for the control terminal associated with the process group of that process, if any. It is useful for programs or shell sequences that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

Files `/dev/tty`

`/dev/tty*`

See Also `ports(1M)`, `console(7D)`

Name ttymux – Serial I/O multiplexing STREAMS device driver

Synopsis

```
multiplexer@0,0:input

multiplexer@0,0:output
```

Description `ttymux` is a STREAMS multiplexer driver that connects multiple serial devices to the system console. Using this driver, input from multiple physical devices can be multiplexed onto a single input stream for the system console. Output written to the console can be distributed to multiple physical devices to provide redundant console interfaces to a system. Input and output can be multiplexed to or from a separate list of devices.

`ttymux` is a STREAMS multiplexer for serial drivers (such as `se(7D)`) that comply with the Solaris terminal subsystem interface.

Currently, multiplexer interfaces are provided for system console I/O only and not for general serial I/O multiplexing. Multiplexer interfaces are currently not available for all platforms. Please see NOTES.

Files `/kernel/drv/sparcv9/ttymux` 64– bit ELF kernel module

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC (NetraCT series only)

See Also `se(7D)`, `termio(7I)`

Writing Device Drivers

Notes Successful loading of this driver and its services depends on the EEPROM or NVRAM settings in effect at the most recent system reboot. Without the platform firmware support, this feature cannot be enabled. Currently, this support is provided only on a NetraCT product family.

Use caution when enabling this feature to perform console input multiplexing, particularly during super-user login. Because no security measures are enabled when the driver is in operation, you must clearly understand the security implications involved in using this feature and take appropriate measures to provide maximum protection to the host. This can include such steps as enabling input to physically secured console devices only.

The `ttymux` driver does not handle the behavioral differences in control characteristics of different terminal types (for example, an ESCAPE sequence.) As a result, multiple terminal types are not supported simultaneously. Please refer to the platform user guide for more information.

Name tzmon – ACPI Thermal Zone Monitor

Description The `tzmon` is a pseudo driver that serves as an ACPI thermal zone monitor. Thermal zones are logical regions within a computer system for which ACPI performs temperature monitoring and control functions. The number of thermal zones on a system with ACPI support varies. For example, some systems may have one or more thermal zones, while others may have none. See the *Advanced Configuration and Power Interface Specification, (ACPI) Version 3.0A*. for more details.

The `tzmon` handles thermal Zone events from ACPI and polls the temperature for each zone exposed by the ACPI implementation. If threshold temperatures are reached, `tzmon` takes appropriate action. For example, if the temperature is sufficiently high and the ACPI implementation supports it, `tzmon` initiates system shutdown.

Note that by default, system temperature control functions are usually performed by the BIOS and may supersede `tzmon` functions, depending on the BIOS implementation. Also, many ACPI implementations expose no thermal zones and in these cases, `tzmon` performs no functions.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWckr
Architecture	x86/x64 only
Interface stability	Private

See Also `attributes(5)`

Advanced Configuration and Power Interface Specification, (ACPI), Version 3.0A.

Name uata – IDE Host Bus Adapter Driver

Synopsis `ide@`*unit-address*

Description The `uata` host bus adapter driver is a nexus driver that supports the `ide` interface on SPARC platforms. The driver attempts to set the disk and ATAPI CD-ROM drive to maximum supported speed. The `uata` driver supports ultra DMA mode-4 (ATA66).

Currently, the `uata` driver supports the CMD646U, Sil680a and Acer Southbridge M5229 IDE controllers. The `uata` driver supports two channels concurrently with two devices connected per channel. The devices are logically numbered from 0 to 3:

0 Master disk on primary channel.

1 Slave disk on primary channel.

2 Master disk on secondary channel.

3 Slave disk on secondary channel.

For ATAPI devices, an ATAPI DRIVE RESET command is issued to facilitate recovery from timeouts and errors. The BSY bit of the drive's status register is polled to check for the drive reset completion. If the drive reset fails, a warning message is displayed and the recovery process continues. This logic is subject to change.

To control the maximum time spent waiting for the ATAPI drive reset to complete, the `atapi-device-reset-waittime` tunable property is available through the `/kernel/drv/uata.conf` file. The default and maximum/minimum values are shown below. Please see `/kernel/drv/uata.conf` for more info.

```
Default value:    3000000
Minimum value:    20
Maximum value:    3000000
```

The `atapi-device-reset-waittime` property units are in microseconds.

Files

/kernel/drv/uata	32–bit ELF kernel module.
/kernel/drv/uata.conf	Driver configuration file.

See Also `prtconf(1M)`, `driver.conf(4)`, attributes(5)

Writing Device Drivers

X3T10 ATA-4 specifications

Diagnostics In addition to being logged, the following messages may appear on the system console:

`ddi_get_iblock_cookie failed.`
The driver could not obtain the interrupt cookie. The attach may fail.

`Drive not ready before set_features.`
Indicates a fatal problem. The drives are not ready to be programmed and features cannot be set. (During the driver initialization process, the driver must set the features for the drive, including `dma` and `pio`).

`Error set after issuing Set Feature command.`
Indicates a fatal problem. The driver's error bit was set after the set feature command was issued. (During the driver initialization process, the driver must set the features for the drive, including dma and pio).

`Interrupt not seen after set_features.`
Indicates a fatal problem with the drive. Features cannot be set.

`ata_controller - set features failed.`
Indicates a fatal problem with the drive. Features cannot be set.

`? target <`*number*`> lun 0.`
Displayed at boot up time to indicate that the target <*number*> was identified, where <*number*> is a decimal value.

`resid`
Residual number of bytes in data transfer and the I/O operation could not be finished completely.

`ghd_timer_newstate: HBA reset failed.`
Generally indicates a fatal condition. I/O operation cannot be completed following reset of the channel.

`timeout: <message> chno =<`*number*`> target=<`*number*`>.`
A timeout occured because of `<message>` on device (`target=<`*number*`>`) on channel (`chno =<`*number*). Where `<message>` could be either early abort, early timeout, abort request, abort device, reset target or reset bus.

`ata_controller - Drive not ready before command <`*number*`>.`
The drive did not respond before issuing the command <*number*> to the controller; command <*number*> will not be issued to the drive. (<*number*> is the hexadecimal opcode for the sleep or standby commands, which are issued when the drive transitions between power management states).

`ATAPI drive reset failed for target: <`*number*`>;Continuing the recovery process.`
If this message is displayed after you modify `/kernel/drv/uata.conf`, try to increase the `atapi-device-reset-waittime` property value within the maximum value allowed, otherwise contact Sun support.

`ata_controller - Command <`*number*`> failed.`
Command <*number*> failed on the drive. (<*number*> is the hexadecimal opcode for the sleep or standby commands, which are issued when the drive transitions between power management states).

`ata_controller - Command <number> returned error.`
The command returned an error. (<*number*> is the hexadecimal opcode for the sleep or standby commands, which are issued when the drive transitions between power management states).

`ata_controller - Cannot take drive <number> to sleep.`
The disk will not transition to sleep state. (Indicates that the driver could not set the device to sleep mode while performing power management functions).

`ata_controller - Cannot reset secondary/primary channel.`
The disk will not transition from sleep to active state.

ata_controller - Unsupported Controller Vendor 0x13d0, Device 0x43f1, Revision 0x034.
An unsupported `ata` controller was found on the system and prints <*ID*>, device id and revision of the controller, where <*ID*> represents the hexidecimal vendor ID.

`Changing the mode of targ: <number> to Ultra DMA mode: <number>.`
For the `timedout` command, the driver attempts to recover by changing speed to lower values and retrying the command. This message indicates to which mode the driver is attempting to re-program the drive, where <*number*> is a decimal value.

`Changing the mode of targ: <number> to Multi DMA mode: <number>.`
For the `timedout` command, the driver attempts to recover by changing speed to lower values and retrying the command. This message indicates to which mode the driver is attempting to re-program the drive, where <*number*> is a decimal value.

These messages are informational and indicate that a timeout occured for a I/O request. The `uata` driver recovers from these states automatically unless there is a fatal error.

Name uath – Atheros AR5523 USB IEEE802.11a/b/g Wireless Network Driver

Description The `uath` IEEE802.11a/b/g wireless network driver is a multi-threaded, loadable, clonable, GLDv3-based STREAMS driver supporting Atheros AR5523 USB IEEE802.11a/b/g wireless network driver.

Configuration The `uath` driver performs auto-negotiation to determine the data rate and mode. The driver supports only BSS networks (also known as `ap` or `infrastructure` networks) and `open` (or `open-system`) or shared system authentication. For wireless security, WEP encryption, WPA-PSk, and WPA2-PSK are currently supported. You can perform configuration and administration tasks using the `dladm`(1M) and `wificonfig`(1M) utilities.

Files

`/dev/uath`	Special character device
`/kernel/drv/uath`	32-bit ELF 86 kernel module (x86)
`/kernel/drv/amd64/uat`	64-bit ELF kernel module (x86)
`/kernel/misc/uathfw`	32-bit ELF firmware kernel module (x86)
`/kernel/misc/amd64/uathfw`	64-bit ELF firmware kernel module (x86)

Attributes See `attributes`(5) for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWuath
Interface Stability	Committed

See Also `dladm`(1M), `wificonfig`(1M), `attributes`(5), `dlpi`(7P), `gld`(7D)

IEEE802.11b/g - Wireless LAN Standard - IEEE, 2003

Name udfs – universal disk format file system

Description The `udfs` file system is a file system type that allows user access to files on Universal Disk Format (UDF) disks from within the Solaris operating environment. Once mounted, a `udfs` file system provides standard Solaris file system operations and semantics. That is, users can read files, write files, and list files in a directory on a UDF device and applications can use standard UNIX system calls on these files and directories.

Because `udfs` is a platform-independent file system, the same media can be written to and read from by any operating system or vendor.

Mounting File Systems `udfs` file systems are mounted using:

```
mount-F udfs -o rw/ro device-special
```

Use:

```
mount /udfs
```

if the `/udfs` and device special file `/dev/dsk/c0t6d0s0` are valid and the following line (or similar line) appears in your `/etc/vfstab` file:

```
/dev/dsk/c0t6d0s0 - /udfs udfs - no ro
```

The `udfs` file system provides read-only support for ROM, RAM, and sequentially–recordable media and read-write support on RAM media.

The `udfs` file system also supports regular files, directories, and symbolic links, as well as device nodes such as block, character, FIFO, and Socket.

See Also `mount(1M)`, `mount_udfs(1M)`, `vfstab(4)`

Notes Invalid characters such as "`NULL`" and "`/`" and invalid file names such as "`.`" and "`..`" will be translated according to the following rule:

Replace the invalid character with an "_," then append the file name with # followed by a 4 digit hex representation of the 16-bit CRC of the original `FileIdentifier`. For example, the file name ".." will become "__#4C05"

Name udp, UDP – Internet User Datagram Protocol

Synopsis

```
#include <sys/socket.h>

#include <netinet/in.h>

s = socket(AF_INET, SOCK_DGRAM, 0);

s = socket(AF_INET6, SOCK_DGRAM, 0);

t = t_open("/dev/udp", O_RDWR);

t = t_open("/dev/udp6", O_RDWR);
```

Description UDP is a simple datagram protocol which is layered directly above the Internet Protocol ("IP") or the Internet Protocol Version 6 ("IPv6"). Programs may access UDP using the socket interface, where it supports the `SOCK_DGRAM` socket type, or using the Transport Level Interface ("TLI"), where it supports the connectionless (`T_CLTS`) service type.

Within the socket interface, UDP is normally used with the `sendto()`, `sendmsg()`, `recvfrom()`, and `recvmsg()` calls (see `send(3SOCKET)` and `recv(3SOCKET)`). If the `connect(3SOCKET)` call is used to fix the destination for future packets, then the `recv(3SOCKET)` or `read(2)` and `send(3SOCKET)` or `write(2)` calls may be used.

UDP address formats are identical to those used by the Transmission Control Protocol ("TCP"). Like TCP, UDP uses a port number along with an IPor IPv6 address to identify the endpoint of communication. The UDP port number space is separate from the TCP port number space, that is, a UDP port may not be "connected" to a TCP port. The `bind(3SOCKET)` call can be used to set the local address and port number of a UDP socket. The local IP or IPv6 address may be left unspecified in the `bind()` call by using the special value `INADDR_ANY` for IP, or the unspecified address (all zeroes) for IPv6. If the `bind()` call is not done, a local IP or IPv6 address and port number will be assigned to the endpoint when the first packet is sent. Broadcast packets may be sent, assuming the underlying network supports this, by using a reserved "broadcast address" This address is network interface dependent. Broadcasts may only be sent by the privileged user.

Note that no two UDP sockets can be bound to the same port unless the bound IP addresses are different. IPv4 `INADDR_ANY` and IPv6 unspecified addresses compare as equal to any IPv4 or IPv6 address. For example, if a socket is bound to `INADDR_ANY` or unspecified address and port X, no other socket can bind to port X, regardless of the binding address. This special consideration of `INADDR_ANY` and unspecified address can be changed using the `SO_REUSEADDR` socket option. If `SO_REUSEADDR` is set on a socket doing a bind, IPv4 `INADDR_ANY` and IPv6 unspecified address do not compare as equal to any IP address. This means that as long as the two sockets are not both bound to `INADDR_ANY`/unspecified address or the same IP address, the two sockets can be bound to the same port.

If an application does not want to allow another socket using the `SO_REUSEADDR` option to bind to a port its socket is bound to, the application can set the socket level option `SO_EXCLBIND` on

a socket. The option values of 0 and 1 represent enabling and disabling the option, respectively. Once this option is enabled on a socket, no other socket can be bound to the same port.

IPv6 does not support broadcast addresses; their function is supported by IPv6 multicast addresses.

Options at the IP level may be used with UDP. See `ip(7P)` or `ip6(7P)`. Additionally, there is one UDP-level option of interest to IPsec Key Management applications (see `ipsec(7P)`and `pf_key(7P)`):

UDP_NAT_T_ENDPOINT
: If this boolean option is set, datagrams sent via this socket will have a non-ESP marker inserted between the UDP header and the data. Likewise, inbound packets that match the endpoint's local-port will be demultiplexed between ESP or the endpoint itself if a non-ESP marker is present. This option is only available on IPv4 sockets (AF_INET), and the application must have sufficient privilege to use PF_KEY sockets to also enable this option.

There are a variety of ways that a UDP packet can be lost or corrupted, including a failure of the underlying communication mechanism. UDP implements a checksum over the data portion of the packet. If the checksum of a received packet is in error, the packet will be dropped with no indication given to the user. A queue of received packets is provided for each UDP socket. This queue has a limited capacity. Arriving datagrams which will not fit within its *high-water* capacity are silently discarded.

UDP processes Internet Control Message Protocol ("ICMP") and Internet Control Message Protocol Version 6 ("ICMP6") error messages received in response to UDP packets it has sent. See `icmp(7P)` and `icmp6(7P)`.

ICMP "source quench" messages are ignored. ICMP "destination unreachable," "time exceeded" and "parameter problem" messages disconnect the socket from its peer so that subsequent attempts to send packets using that socket will return an error. UDP will not guarantee that packets are delivered in the order they were sent. As well, duplicate packets may be generated in the communication process.

ICMP6 "destination unreachable" packets are ignored unless the enclosed code indicates that the port is not in use on the target host, in which case, the application is notified. ICMP6 "parameter problem" notifications are similarly passed upstream. All other ICMP6 messages are ignored.

See Also `read(2)`, `write(2)`, `bind(3SOCKET)`, `connect(3SOCKET)`, `recv(3SOCKET)`, `send(3SOCKET)`, `icmp(7P)`, `icmp6(7P)`, `inet(7P)`, `inet6(7P)`, `ip(7P)`, `ipsec(7P)`, `ip6(7P)`, `pf_key(7P)`, `tcp(7P)`

Postel, Jon, *RFC 768, User Datagram Protocol*, Network Information Center, SRI International, Menlo Park, Calif., August 1980

Huttunen, A., Swander, B., Volpe, V., DiBurro, L., Stenberg, *M., RFC 3948, UDP Encapsulation of IPsec ESP Packets*, The Internet Society, 2005.

Diagnostics A socket operation may fail if:

`EISCONN`	A `connect()` operation was attempted on a socket on which a `connect()` operation had already been performed, and the socket could not be successfully disconnected before making the new connection.
`EISCONN`	A `sendto()` or `sendmsg()` operation specifying an address to which the message should be sent was attempted on a socket on which a `connect()` operation had already been performed.
`ENOTCONN`	A `send()` or `write()` operation, or a `sendto()` or `sendmsg()` operation not specifying an address to which the message should be sent, was attempted on a socket on which a `connect()` operation had not already been performed.
`EADDRINUSE`	A `bind()` operation was attempted on a socket with a network address/port pair that has already been bound to another socket.
`EADDRNOTAVAIL`	A `bind()` operation was attempted on a socket with a network address for which no network interface exists.
`EINVAL`	A `sendmsg()` operation with a non-NULL `msg_accrights` was attempted.
`EACCES`	A `bind()` operation was attempted with a "reserved" port number and the effective user ID of the process was not the privileged user.
`ENOBUFS`	The system ran out of memory for internal data structures.

Name ufs – UFS file system

Synopsis

```
#include <sys/param.h>

#include <sys/types.h>

#include <sys/fs/ufs_fs.h>

#include <sys/fs/ufs_inode.h>
```

Description UFS is the default disk-based file system for the Solaris environment. The UFS file system is hierarchical, starting with its root directory (/) and continuing downward through a number of directories. The root of a UFS file system is `inode 2`. A UFS file system's root contents replace the contents of the directory upon which it is mounted.

Subsequent sections of this manpage provide details of the UFS file systems.

State Flags (`fs_state` and `fs_clean`) UFS uses state flags to identify the state of the file system. `fs_state` is `FSOKAY - fs_time`. `fs_time` is the timestamp that indicates when the last system write occurred. `fs_state` is updated whenever `fs_clean` changes. Some `fs_clean` values are:

`FSCLEAN` Indicates an undamaged, cleanly unmounted file system.

`FSACTIVE` Indicates a mounted file system that has modified data in memory. A mounted file system with this state flag indicates that user data or metadata would be lost if power to the system is interrupted.

`FSSTABLE` Indicates an idle mounted file system. A mounted file system with this state flag indicates that neither user data nor metadata would be lost if power to the system is interrupted.

`FSBAD` Indicates that this file system contains inconsistent file system data.

FSLOG Indicates that the file system has logging enabled. A file system with this flag set is either mounted or unmounted. If a file system has logging enabled, the only flags that it can have are `FSLOG` or `FSBAD`. A non-logging file system can have `FSACTIVE`, `FSSTABLE`, or `FSCLEAN`.

It is not necessary to run the `fsck` command on unmounted file systems with a state of `FSCLEAN`, `FSSTABLE`, or `FSLOG`. `mount(2)` returns `ENOSPC` if an attempt is made to mount a UFS file system with a state of `FSACTIVE` for read/write access.

As an additional safeguard, `fs_clean` should be trusted only if `fs_state` contains a value equal to `FSOKAY - fs_time`, where `FSOKAY` is a constant integer defined in the `/usr/include/sys/fs/ufs_fs.h` file. Otherwise, `fs_clean` is treated as though it contains the value of `FSACTIVE`.

Extended Fundamental Types (EFT) Extended Fundamental Types (EFT) provide 32-bit user ID (UID), group ID (GID), and device numbers.

If a UID or GID contains an extended value, the short variable (`ic_suid`, `ic_sgid`) contains the value 65535 and the corresponding UID or GID is in `ic_uid` or `ic_gid`. Because numbers for block and character devices are stored in the first direct block pointer of the inode (`ic_db[0]`) and the disk block addresses are already 32 bit values, no special encoding exists for device numbers (unlike UID or GID fields).

Multiterabyte File System

A multiterabyte file system enables creation of a UFS file system up to approximately 16 terabytes of usable space, minus approximately one percent overhead. A sparse file can have a logical size of one terabyte. However, the actual amount of data that can be stored in a file is approximately one percent less than one terabyte because of file system overhead.

On-disk format changes for a multiterabyte UFS file system include:

- The magic number in the superblock changes from `FS_MAGIC` to `MTB_UFS_MAGIC`. For more information, see the `/usr/include/sys/fs/ufs_fs` file.
- The `fs_logbno` unit is a sector for UFS that is less than 1 terabyte in size and fragments for a multiterabyte UFS file system.

UFS Logging

UFS logging bundles the multiple metadata changes that comprise a complete UFS operation into a transaction. Sets of transactions are recorded in an on-disk log and are applied to the actual UFS file system's metadata.

UFS logging provides two advantages:

1. A file system that is consistent with the transaction log eliminates the need to run `fsck` after a system crash or an unclean shutdown.
2. UFS logging often provides a significant performance improvement. This is because a file system with logging enabled converts multiple updates to the same data into single updates, thereby reducing the number of overhead disk operations.

The UFS log is allocated from free blocks on the file system and is sized at approximately 1 Mbyte per 1 Gbyte of file system, up to 256 Mbytes. The log size may be larger (up to a maximum of 512 Mbytes), depending upon the number of cylinder groups present in the file system. The log is continually flushed as it fills up. The log is also flushed when the file system is unmounted or as a result of a `lockfs(1M)` command.

Mounting UFS File Systems

You can mount a UFS file system in various ways using syntax similar to the following:

1. Use `mount` from the command line:

```
# mount -F ufs /dev/dsk/c0t0d0s7 /export/home
```

2. Include an entry in the `/etc/vfstab` file to mount the file system at boot time:

```
/dev/dsk/c0t0d0s7 /dev/rdsk/c0t0d0s7 /export/home  ufs   2   yes  -
```

For more information on mounting UFS file systems, see `mount_ufs(1M)`.

Attributes See attributes(5) for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Uncommitted

See Also df(1M), fsck(1M), fsck_ufs(1M), fstyp(1M), lockfs(1M), mkfs_ufs(1M), newfs(1M), ufsdump(1M), ufsrestore(1M), tunefs(1M), mount(2), attributes(5)

Writing Device Drivers

Notes For information about internal UFS structures, see newfs(1M) and mkfs_ufs(1M). For information about the ufsdump and ufsrestore commands, see ufsdump(1M), ufsrestore(1M), and /usr/include/protocols/dumprestore.h.

If you experience difficulty in allocating space on the ufs filesystem, it may be due to framentation. Fragmentation can occur when you do not have sufficient free blocks to satisfy an allocation request even though df(1M) indicates that enough free space is available. (This may occur because df only uses the available fragment count to calculate available space, but the file system requires contiguous sets of fragments for most allocations). If you suspect that you have exhausted contiguous fragments on your file system, you can use the fstyp(1M) utility with the -v option. In the fstyp output, look at the *nbfree* (number of blocks free) and *nffree* (number of fragments free) fields. On unmounted filesystems, you can use fsck(1M) and observe the last line of output, which reports, among other items, the number of fragments and the degree of fragmentation. To correct a fragmentation problem, run ufsdump(1M) and ufsrestore(1M) on the ufs filesystem.

Name ugen – USB generic driver

Synopsis

```
Node Name@unit-address

#include <sys/usb/clients/ugen/usb_ugen.h>
```

Description ugen is a generic USBA (Solaris USB Architecture) compliant client character driver that presents USB devices to applications through a standard open(2), close(2), read(2), write(2), aioread(3C), aiowrite(3C) Unix interface. Uninterpreted raw data are transfered to and from the device via file descriptors created for each USB endpoint. Status is obtained by reading file descriptors created for endpoint and full device status.

ugen supports control, bulk, isochronous and interrupt (in and out) transfers. libusb(3LIB) uses ugen to access devices that do not contain drivers (such as digital cameras and PDAs). Refer to /usr/sfw/share/doc/libusb/libusb.txt for details.

Binding In general, no explicit binding of the ugen driver is necessary because usb_mid(7D) is the default driver for devices without a class or vendor unique driver. usb_mid(7D) creates the same logical device names as ugen, but only if no child interfaces are explicitly bound to ugen. If it is necessary to bind ugen explicitly to a device or interface, the following section explains the necessary steps.

ugen can bind to a device with one or more interfaces in its entirety, or to a single interface of that device. The binding type depends on information that is passed to add_drv(1M) or update_drv(1M).

An add_drv(1M) command binds ugen to a list of device types it is to control. update_drv(1M) adds an additional device type to the list of device types being managed by the driver.

Names used to bind drivers can be found in /var/adm/messages. When a device is on-lined after hot insertion, and no driver is found, there will be an entry containing:

```
USB 2.0 device (usb<vid>,<pid>)...
```

where vid is the USB vendor identifier in hex and pid is the product identifier in hex supplied by the device descriptor usb_dev_descr(9S).

When using ugen for the first time, you must add the driver utilizing add_drv(1M), using a command of the following form:

```
Assuming that the vid is 472 and pid is b0b0:

add_drv -n -m '* <device perms> <owner> <group>'
   -i '"usb472,b0b0"' ugen
```

If the command fails with:

```
(ugen) already in use as a driver or alias.
```

...add the device using update_drv(1M):

```
update_drv -a -m '* <device perms> <owner> <group>'
   -i '"usb472,b0b0"' ugen
```

This binds ugen to the entire device.

If ugen only binds to one interface of the device, use the following driver_alias instead of usb<vid>,<pid>:

```
usbif<vid>,<pid>.config<cfg value>.<interface number>
```

where cfg value is the value of bConfigurationValue in the configuration descriptor (usb_cfg_descr(9S)). For example "usbif1234,4567.config1.0."

Note that you can use update_drv to also remove bindings. Please see update_drv(1M) for more information.

After a successful add_drv or update_drv, remove the device and reinsert. Check with the prtconf(1M) -D option to determine if ugen is successfully bound to the device and the nodes created in /dev/usb/<vid>.<pid> (see below).

An example showing how to bind a child device representing interface 0 of configuration 1 of a composite device follows:

```
update_drv -a -m '* 0666 root sys'
    -i '"usbif472,b0b0.config1.0"' ugen
```

Note that you can completely uninstall the ugen driver and delete it from the system by doing:

```
pkgrm SUNWugen
```

Any pkgadd of SUNWugen after the pkgrm reactivates any pre-existing ugen driver device-bindings.

Any pre-existing ugen driver device-bindings are preserved across operating system upgrades.

Logical Device Name Format

For each device or child device it manages, ugen creates one logical device name for device-wide status and one logical device name for endpoint 0. ugen also creates logical device names for all other endpoints within the device node's binding scope (interface or device), plus logical device names for their status.

If separate ugen instances control different interfaces of the same device, the device-wide status and endpoint logical device names created for each instance will share access to the same source or endpoint pipes. For example, a device with two interfaces, each operated by their own ugen instance, will show `endpoint0` as `if0cntrl0` to the first interface, and will show it as `if1cntrl0` to the second interface. Both of these logical device names share `endpoint0`. Likewise for the same device, ugen makes the device-wide status available as `if0devstat` to the first interface and as `if1devstat` to the second interface. `if0devstat` and `if1devstat` both return the same data.

Any ugen logical device name can be held open by only one user at a time, regardless of whether the O_EXCL flag passed to open(2). When a single pipe or data source is shared by multiple logical device names, such as if[0,1]cntrl0 or if[0,1]devstat above, more than one logical device name sharing the pipe or data source can be open at a time. However, only one user may access the shared pipe or data source at a time, regardless of the logical device name used for access.

When ugen is bound to an entire device, the following logical device names are created (each on a single line). *N* represents the instance number of the device type.

```
Endpoint 0 (default endpoint):

        /dev/usb/<vid>.<pid>/<N>/cntrl0
        /dev/usb/<vid>.<pid>/<N>/cntrl0stat

    For example:

        /dev/usb/472.b0b0/0/cntrl0
        /dev/usb/472.b0b0/0/cntrl0stat

Configuration index 1, Endpoints > 0, alternate 0:

        /dev/usb/<vid>.<pid>/<N>/if<interface#>
                                <in|out|cntrl><endpoint#>
        /dev/usb/<vid>.<pid>/<N>/if<interface#>
                                <in|out|cntrl><endpoint#>stat

    For example:

        /dev/usb/472.b0b0/0/if0in1
        /dev/usb/472.b0b0/0/if0in1stat

Configuration index 1, Endpoints > 0, alternate > 0:

        /dev/usb/<vid>.<pid>/<N>/if<interface#>.
                                <alternate><in|out|cntrl><endpoint#>
        /dev/usb/<vid>.<pid>/<N>/if<interface#>.
                                <alternate<in|out|cntrl><endpoint#>stat

    For example:

        /dev/usb/472.b0b0/0/if0.1in3
        /dev/usb/472.b0b0/0/if0.1in3stat

Configuration index> 1, Endpoints > 0, alternate 0:
        /dev/usb/<vid>.<pid>/<N>/cfg<value>if<interface#>
                                <in|out|cntrl><endpoint#>
        /dev/usb/<vid>.<pid>/<N>/cfg<value>if<interface#>
```

```
                                <in|out|cntrl><endpoint#>stat

    For example:

       /dev/usb/472.b0b0/0/cfg2if0in1
       /dev/usb/472.b0b0/0/cfg2if0in1stat

    Note that the configuration value from the configuration
    descriptor indexed by the configuration index is used in
    the node name and not the configuration index itself.

Configuration index> 1, Endpoints > 0, alternate > 0:
        /dev/usb/<vid>.<pid>/<N>/cfg<value>if<interface#>.
                                <alternate<in|out|cntrl><endpoint#>
        /dev/usb/<vid>.<pid>/<N>/cfg<value>if<interface#>.
                                <alternate<in|out|cntrl><endpoint#>stat

    For example:

        /dev/usb/472.b0b0/0/cfg2if0.1in1
        /dev/usb/472.b0b0/0/cfg2if0.1in1stat

  Device status:

        /dev/usb/<vid>.<pid>/<N>/devstat

    For example:

        /dev/usb/472.b0b0/0/devstat
```

When ugen is bound to a single device interface, the following logical device nodes are created:

```
Endpoint 0 (default endpoint):

         /dev/usb/<vid>.<pid>/<N>/if<interface#>cntrl0
         /dev/usb/<vid>.<pid>/<N>/if<interface#>cntrl0stat

    For example:

         /dev/usb/472.b0b0/0/if0cntrl0
         /dev/usb/472.b0b0/0/if0cntrl0stat

Device status:
        /dev/usb/<vid>.<pid>/<N>/if<interface#>devstat

    For example:
        /dev/usb/472.b0b0/0/if0devstat
```

The format for all other logical device names is identical to the format used when ugen is bound to the entire device.

Opening the endpoint of a different configuration or different alternate interface will cause an implicit change of configuration or a switch to an alternate interface. A configuration change is prohibited when any non-zero endpoint device nodes are open. An alternate interface switch is prohibited if any endpoint in the same interface is open.

Hot-PLUGGING A device may be hot-removed at any time. Following hot-removal, the device status changes to USB_DEV_STAT_DISCONNECTED, the status of open endpoints change to USB_LC_STAT_DISCONNECTED upon their access, and all subsequent transfer requests fail. Endpoints are reactivated by first reinserting the device and then closing and reopening all endpoints that were open when the device was disconnected.

Cpr (CHECKPOINT/RESUME) CPR (Checkpoint/Resume) may be initiated at any time and is treated similarly to a hot-removal. Upon successful suspend and resume, all subsequent transfer requests fail as an indication to the application to reinitialize. Applications should close and reopen all endpoints to reinstate them. All endpoint and device status on Resume (before close and reopen) is USB_LC_STAT_SUSPENDED. A system suspend will fail while ugen is performing a transfer.

Device Power Management Devices which support remote wakeup can be power managed when they have no open logical device nodes. When an application opens the first logical device node of a device, that application should assume that a reinitialization of device state is required.

Device Status Management Applications can monitor device status changes by reading the device status from the device status logical name. When opened without O_NONBLOCK and O_NDELAY, all reads from that file descriptor (with the exception of the the intial read that follows the open) block until a device status change occurs. Calls to read will always return immediately if opened with `O_NONBLOCK` or `O_NDELAY`. Nonblocking calls to read which have no data to return, return no error and zero bytes read.

Device statuses are:

USB_DEV_STAT_ONLINE	Device is available.
USB_DEV_STAT_DISCONNECTED	Device has been disconnected.
USB_DEV_STAT_RESUMED	Device has been resumed, however, endpoints which were open on suspend have not yet been closed and reopened.
USB_DEV_STAT_UNAVAILABLE	Device has been reconnected, however, endpoints which were open on disconnect have not yet been closed and reopened.

The following code reads the device status device logical name:

```
int fd;
int status;

if ((fd = open("/dev/usb/472.b0b0/0/devstat",
    O_RDONLY)) < 0)     {
        /* handle error */
}

if (read(fd, &status, sizeof(status))  != sizeof(status)) {
        /* handle error */
}

switch (status) {
case USB_DEV_STAT_DISCONNECTED:
        printf ("Terminating as device has been disconnected.\n");
        exit (0);

case USB_DEV_STAT_RESUMED:
case USB_DEV_STAT_UNAVAILABLE:
        /*
         * Close and reopen endpoints to reestablish device access,
         * then reset device.
         */
        break;

case USB_DEV_STAT_ONLINE:
default:
        break;
}
```

Use `poll(2)` to block on several logical names simultaneously, including device status logical names. Poll indicates when reading a logical name would return data. See `poll(2)` for details. Calls to read may be done whether or not they follow calls to poll.

Endpoint Status Management

Each data endpoint has a corresponding status logical name. Use the status logical name to retrieve the state of the data endpoint, including detail on how its most recent transfer failed. Reads of the status file descriptors always return immediately. See the ERRORS section for more information on endpoint status values. All logical device name files created for returning status must be opened with `O_RDONLY`.

The following code illustrates reading the status file descriptor of an endpoint which just failed a data transfer in order to get more information on the failure.

```
int data_xfered, status;
int ep1_data_fd, ep1_stat_fd;
uchar_t request[8];

ep1_data_fd = open ("/dev/usb/472.b0b0/0/if0out1", O_WRONLY);
```

```
if (ep1_data_fd < 0) {
        /* Handle open error. */
}

ep1_stat_fd = open ("/dev/usb/472.b0b0/0/if0out1stat",
    O_RDONLY);
if (ep1_stat_fd < 0) {
        /* Handle open error. */
}

data_xfered = write(ep1_data_fd, request, sizeof (request));

/* An error occured during the data transfer. */
if (data_xfered != sizeof (request)) {

        /* Read status file descriptor for details on failure. */
        if (read(ep1_stat_fd, (int *)&status, sizeof (status)) !=
            sizeof (status)) {
                status = USB_LC_STAT_UNSPECIFIED_ERR;
        }

        /* Take appropriate action. */
        switch (status) {
        case USB_LC_STAT_STALL:
                printf ("Endpoint stalled.\n");
                break;
        case ...
                ...
        }

   }
```

Control Transfers The control endpoint is typically used to set up the device and to query device status or configuration.

Applications requiring I/O on a control endpoint should open the corresponding logical device name and use regular UNIX I/O system calls. For example: `read(2)`, `write(2)`, `aioread(3C)` and `aiowrite(3C)`. `poll(2)` is not supported on control endpoints.

A control endpoint must be opened with `O_RDWR` since it is bidirectional. It cannot be opened with `O_NONBLOCK` or `O_NDELAY`.

For example:

```
fd = open("/dev/usb/472.b0b0/0/cntrl0", O_RDWR);
```

```
fdstat = open("/dev/usb/472.b0b0/0/cntrl0stat", O_RDONLY);
```

Control endpoints can be read and written. A `read` operation receives data *from* the device and a `write` operation sends data *to* the device.

To perform a control-IN transfer, perform a `write(2)` of USB setup data (see section 9.3 of the *USB 1.1* or *2.0* specifications) followed by a `read(2)` on the same control endpoint to fetch the desired data. For example:

```
void init_cntrl_req(
    uchar_t *req, uchar_t bmRequestType, uchar_t bRequest,
    ushort_t wValue, ushort_t wIndex, ushort_t wLength) {
        req[0] = bmRequestType;
        req[1] = bRequest;
        req[2] = 0xFF & wValue;
        req[3] = 0xFF & (wValue >> 8);
        req[4] = 0xFF & wIndex;
        req[5] = 0xFF & (wIndex >> 8);
        req[6] = 0xFF & wLength;
        req[7] = 0xFF & (wLength >> 8);
 }

 ....


        uchar_t dev_descr_req[8];
        usb_dev_descr_t descr;

        init_cntrl_req(dev_descr_req,
            USB_DEV_REQ_DEV_TO_HOST, USB_REQ_GET_DESCR,
            USB_DESCR_TYPE_SETUP_DEV, 0, sizeof (descr));

        count = write(fd, dev_descr_req, sizeof (dev_descr_req));
        if (count != sizeof (dev_descr_req)) {
                /* do some error recovery */
                ...
        }

        count = read(fd, &descr, sizeof (descr));
        if (count != sizeof (descr)) {
                /* do some error recovery */
        }
```

The application can issue any number of reads to read data received on a control endpoint. ugen successfully completes all reads, returning the number of bytes transferred. Zero is returned when there is no data to transfer.

If the read/write fails and returns –1, you can access the endpoint's status device logical name for precise error information:

```
int status;

count = read(fdstat, &status, sizeof (status));
if (count == sizeof (status)) {
        switch (status) {
        case USB_LC_STAT_SUSPENDED:
        case USB_LC_STAT_DISCONNECTED:
                /* close all endpoints */
                ...
                break;
        default:
                ...
                break;
        }
}
```

Refer to the ERRORS section for all possible error values.

To perform a control-OUT transfer, send in a single transfer, the USB setup data followed by any accompanying data bytes.

```
/* 1st 8 bytes of wbuf are setup. */
init_cntrl_req(wbuf, .......);

/* Data bytes begin at byte 8 of wbuf. */
bcopy(data, &wuf[8], sizeof (data));

/* Send it all in a single transfer. */
count = write(fd, wbuf, sizeof (wbuf));
```

A write(2) returns the number of bytes (both setup and data) actually transferred, (whether or not the write is completely successful), provided that some data is actually transferred. When no data is transferred, write(2) returns -1. Applications can read the corresponding endpoint status to retrieve detailed error information. Note that it is an error to specify a size different than:

(number of data bytes + number of setup bytes).

Here is a more extensive example which gets all descriptors of a device configuration. For sake of brevity, uninteresting parts are omitted.

```
#include <sys/usb/usba.h>
#include <sys/usb/clients/ugen/usb_ugen.h>

uchar_t *config_cloud;
uchar_t *curr_descr;
```

```
uchar_t *bytes;

int curr_descr_len;
int curr_descr_type;

usb_cfg_descr_t cfg_descr;
usb_if_descr_t if_descr;
usb_ep_descr_t ep_descr;

/* See 9.13 of USB 2.0 spec for ordering. */
static char *pipetypes[] = {
     "Control", "Isochronous", "Bulk", "Interrupt"
};

/*
 * Setup to send a request to read just the config descriptor.  The
 * size of the whole cloud, containing all cfg, interface, endpoint,
 * class and vendor-specific descriptors, will be returned as part of
 * the config descriptor.
 */
init_cntrl_req(&setup_data, USB_DEV_REQ_DEV_TO_HOST, USB_REQ_GET_DESCR,
                USB_DESCR_TYPE_SETUP_CFG, 0, USB_CFG_DESCR_SIZE);

/*
 * Write setup data. USB device will prepare to return the whole
 * config cloud as a response to this. We will read this separately.
 */
count = write(ctrl_fd, &setup_data, sizeof (setup_data));
if (count != sizeof (setup_data)) {
         /* Error recovery. */
} else {
        count = read(ctrl_fd, &cfg_descr, USB_CFG_DESCR_SIZE);
        if (count != USB_CFG_DESCR_SIZE) {
                 /* Error recovery. */
        }
}

/* USB data is little endian. */
bytes = (uchar_t *)(&cfg_descr.wTotalLength);
totalLength = bytes[0] + (bytes[1] << 8);

/*
 * The size of the whole cloud is in the bLength field.  Set up
 * to read this amount of data, to get the whole cloud.
 */
config_cloud = malloc(totalLength);
```

```
init_cntrl_req(&setup_data, USB_DEV_REQ_DEV_TO_HOST, USB_REQ_GET_DESCR,
                USB_DESCR_TYPE_SETUP_CFG, 0, totalLength);

count = write(ctrl_fd, &setup_data, sizeof (setup_data));
if (count != sizeof (setup_data)) {
        /* Error recovery. */
} else {
        count = read(ctrl_fd, config_cloud, totalLength);
        if (count != totalLength) {
                /* Error recovery. */
         }
}

/* Got the data.  Now loop, dumping out the descriptors found. */

curr_descr = config_cloud;
offset = 0;
while (offset < totalLength) {

        /* All descr have length and type at offset 0 and 1 */
        curr_descr_len = curr_descr[0];
        curr_descr_type = curr_descr[1];

        switch (curr_descr_type) {
        case USB_DESCR_TYPE_CFG:

                /*
                 * Copy data into separate structure, needed for
                 * proper alignment of all non char fields.  Note:
                 * non-char fields of all descriptors begin on aligned
                 * boundaries.  The issue is that some structures may
                 * be adjacent to others which have an odd-numbered
                 * byte size, and may thus start on an odd-numbered
                 * boundary.  */
                bcopy(curr_descr, &cfg_descr, curr_descr_len);

                /* Remember to read any words in endian-neutral way. */

                (void) printf("\nConfig %d found.\n",
                    cfg_descr.bConfigurationValue);
                break;

         case USB_DESCR_TYPE_IF:
                bcopy(curr_descr, &if_descr, curr_descr_len);
                (void) printf("\n\tInterface %d, Alt %d found.\n",
                    if_descr.bInterfaceNumber,
```

```
                        if_descr.bAlternateSetting);
                    break;

                case USB_DESCR_TYPE_EP:
                    bcopy(curr_descr, &ep_descr, curr_descr_len);
                    (void) printf("\n\t\tEndpoint %d (%s-%s) found.\n",
                        (ep_descr.bEndpointAddress & USB_EP_NUM_MASK),
                        (pipetypes[
                            ep_descr.bmAttributes & USB_EP_ATTR_MASK]),
                        ((ep_descr.bEndpointAddress &
                        USB_EP_DIR_IN) ? "IN" : "OUT"));
                     break;

                 default:
                     (void) printf(
                         "\n\t\t\tOther descriptor found.  Type:%d\n",
                         curr_descr_type);
                     break;
                 }

                 offset += curr_descr_len;
                 curr_descr = &config_cloud[offset];
        }
```

Interrupt-IN Transfers

Applications requiring data from an interrupt-IN endpoint should open the corresponding logical device name and use `read(2)`, `aioread(3C)` and `poll(2)` system calls.

An interrupt-IN endpoint must be opened with `O_RDONLY`. It can also be opened using `O_NONBLOCK` or `O_NDELAY` if desired.

```
fd = open("/dev/usb/472.b0b0/0/if0in1", O_RDONLY);
```

```
fdstat = open("/dev/usb/472.b0b0/0/if0in1stat", O_RDONLY);
```

`ugen` starts polling interrupt—IN endpoints immediately upon opening them and stops polling them upon closure. (Polling refers to interrogation of the device by the driver and should not be confused with `poll(2)`, which is an interrogation of the driver by the application.)

A `read(2)` of an endpoint opened with the `O_NONBLOCK` or `O_NDELAY` flags set will not block when there is insufficient data available to satisfy the request. The `read` simply returns what it can without signifying any error.

Applications should continuously check for and consume interrupt data. `ugen` enables buffering of up to one second of incoming data. In case of buffer overflow, `ugen` stops polling the interrupt-IN endpoint until the application consumes all the data. In this case, a `read(2)` of an empty buffer returns `-1`, sets the endpoint status to `USB_LC_STAT_INTR_BUF_FULL` (to

indicate that the buffer had been full and polling had been stopped) and causes ugen to start polling the endpoint again. To retrieve the status, the application can open and read the corresponding endpoint's status device logical name.

```
for (;;) {
        count = read(fd, buf, sizeof(buf));
        if (count == -1) {
                int cnt, status;

                cnt = read(fdstat, &status, sizeof (status));
                if (cnt == -1) {
                         /* more error recovery here */
                } else {
                        switch (status) {
                        case USB_LC_STAT_INTR_BUF_FULL:
                               ...
                               break;
                        default:
                               ...
                               break;
                        }
                }
           }
           /* process the data */
           ....
        }
```

ugen will never drop data. However, the device may drop data if the application cannot read it at the rate that it is produced.

Applications requiring unbuffered data from an interrupt-IN endpoint should open the associated status endpoint with O_RDWR before opening the associated interrupt-IN endpoint and write a control byte with USB_EP_INTR_ONE_XFER set. All other bits are reserved and should be 0.

"One transfer" mode will persist until disabled explicitly after the associated interrupt-IN endpoint has been closed by writing a control byte with USB_EP_INTR_ONE_XFER cleared.

"One transfer" mode is implicitly disabled when the status/control endpoint is closed.

Attempts to change the "one transfer" mode while the endpoint is open will result in EINVAL.

An application can open multiple interrupt-IN endpoints and can call poll(2) to monitor the availability of new data. (Note: poll works with interrupt-IN data endpoints, not their status endpoints.)

```
        struct pollfd pfd[2];
```

```
bzero(pfd, sizeof (pfd));
pfd[0].fd = fd1;    /* fd1 is one interrupt-IN endpoint. */
pfd[0].events = POLLIN;
pfd[1].fd = fd2;    /* fd2 is another interrupt-IN endpoint. */
pfd[1].events = POLLIN;

for (;;) {
        poll(pfd, 2, -1);

        if (pfd[0].revents & POLLIN) {
                count = read(fd1, buf, sizeof (buf));
                ....
        }
        if (pfd[1].revents & POLLIN) {
                count = read(fd2, buf, sizeof (buf));
                ....
        }
}
```

You can monitor the device status endpoint via `poll(2)` concurrently with the multiple interrupt-IN endpoints. Simply add another pollfd element to the pfd array in the previous code example, and initialize the new element's `fd` field with the file descriptor of the device status endpoint (opened without O_NONBLOCK or O_NDELAY). Set the new element's event field to POLLIN like the other elements. Note that only interrupt–IN endpoints and the device status endpoint can be monitored using `poll(2)`.

Interrupt-OUT Transfers

Applications requiring output on an interrupt-OUT endpoint can open the corresponding logical device name and perform regular UNIX I/O system calls such as `write(2)` and `aiowrite(3C)`.

An interrupt-OUT endpoint must be opened with O_WRONLY.

```
fd = open("/dev/usb/472.b0b0/0/if0out3", O_WRONLY);

fdstat = open("/dev/usb/472.b0b0/0/if0out3stat", O_RDONLY);
```

Data can be written to an interrupt-OUT endpoint as follows:

```
count = write(fd, buf, sizeof (buf)):
if (count == -1) {
       /* error recovery */
}
```

Bulk Transfers

Applications requiring I/O on a bulk endpoint can open the corresponding logical device name and perform regular UNIX I/O system calls. For example: `read(2)`, `write(2)`, `aioread(3C)` and `aiowrite(3C)`. `poll(2)` is not supported on bulk endpoints.

A bulk endpoint must be opened with O_RDONLY or O_WRONLY and cannot be opened with O_NONBLOCK or O_NDELAY:

```
fd = open("/dev/usb/472.b0b0/0/if0in2", O_RDONLY);

fdstat = open("/dev/usb/472.b0b0/0/if0in2stat", O_RDONLY);
```

Data can be read from a bulk-IN endpoint as follows:

```
        count = read(fd, buf, sizeof (buf)):
        if (count == -1) {
                /* error recovery */
        }

 Data can be written to a bulk-OUT endpoint as follows:

        count = write(fd, buf, sizeof (buf)):
        if (count == -1) {
                /* error recovery */
        }
```

Isochronous Transfers

Applications requiring I/O on an isochronous endpoint can open the corresponding logical device name and perform regular UNIX I/O system calls such as read(2), write(2), poll(2), aioread(3C) and aiowrite(3C). An isochronous endpoint must be opened with O_RDWR.

```
fd = open("/dev/usb/472.b0b0/0/if0.3in2", O_RDWR);

fdstat = open("/dev/usb/472.b0b0/0/if0.3in2stat", O_RDONLY);
```

Applications can use the status logical name to retrieve the state of the isochronous data endpoint, including details on why the most recent transfer failed.

Applications have the flexibility to specify the number of isochronous packets and the size of individual packets they want to transfer. Applications should use the following data structures to exchange isochronous packet information with the ugen driver:

```
typedef struct ugen_isoc_pkt_descr {
        /*
        * Set by the application, for all isochro.
        * requests, to the num. of bytes to xfer
        * in a packet.
        */
           ushort_t        dsc_isoc_pkt_len;

        /*
        * Set by ugen to actual num. of bytes sent/received
        * in a packet.
            */
       ushort_t        dsc_isoc_pkt_actual_len;
```

```
        /*
        * Per pkt. status set by ugen driver both for the
        * isochronous IN and OUT requests. Application can
        * use USB_LC_STAT_* to parse the status.
        */
      int     dsc_isoc_pkt_status;
} ugen_isoc_pkt_descr_t;

typedef struct ugen_isoc_req_head {
  /* pkt count of the isoc request */
   int req_isoc_pkts_count;

 /* pkt descriptors */
 ugen_isoc_pkt_descr_t req_isoc_pkt_descrs[1];
} ugen_isoc_req_head_t;
```

`req_isoc_pkts_count` is limited by the capability of the USB host controller driver. The current upper bound for the `uhci` and `ohci` drivers is 512. The upper bound for the `ehci` driver is 1024.

For an isochronous-IN endpoint, applications must first use the `ugen_isoc_req_head_t` structure followed by `ugen_isoc_pkt_descr_t` to write packet request information to the `ugen` node. The `ugen` driver then checks the validity of the request. If it is valid, the driver immediately begins isochronous polling on the IN endpoint and applications can proceed with `read(2)` of the data on the isochronous-IN endpoint. Upon successful return of `read(2)`, isochronous packet descriptors (whose `dsc_isoc_pkt_actual_len` and `dsc_isoc_pkt_status` fields were filled by the driver) are returned, followed by the request's device payload data.

Applications should continuously check for and consume isochronous data. The `ugen` driver enables buffering of up to eight seconds of incoming data for full-speed isochronous endpoint, one second of data for high-speed isochronous endpoints who request one transaction per microframe and 1/3 of a second of incoming data for high-speed high-bandwidth isochronous endpoints who request three transactions per microframe. In case of buffer overflow, `ugen` discards the oldest data.

The isochronous-IN polling can only be stopped by a `close(2)` associated file descriptor. If applications want to change packet information, they must first `close(2)` the endpoint to stop the isochronous-IN polling, then `open(2)` the endpoint and `write(2)` new packets request.

The following example shows how to read an isochronous-IN endpoint:

```
        #include  <sys/usb/clients/ugen/usb_ugen.h>

        char *buf, *p;
        ushort_t  pktlen;
```

```
    int pktcnt, i;
    int len;
    ugen_isoc_req_head_t *req;
    ugen_isoc_pkt_descr_t *pktdesc;
    char rdbuf[5000];

    pktcnt = 4; /* 4 packets in this request */

    len = sizeof(int) +
        sizeof(ugen_isoc_pkt_descr_t) * pktcount;

    buf = malloc(len);
    if (!buf) {
        /* Error recovery. */
     }

    req = (ugen_isoc_req_head_t *)buf;
    req->req_isoc_pkts_count = pktcnt;

    pktdesc = (ugen_isoc_pkt_descr_t *)
        (req->req_isoc_pkt_descrs);

for (i = 0; i < pktcnt; i++) {
           /*
           * pktlen should not exceed xfer
           * capability of an endpoint
           */
           pktdesc[i].dsc_isoc_pkt_len = pktlen;

            pktdesc[i].dsc_isoc_pkt_actual_len = 0;
           pktdesc[i].dsc_isoc_pkt_status = 0;
    }

       /*
       * write request info to driver and len must
       * be exactly the sum of
       * sizeof(int) + sizeof(ugen_isoc_pkt_descr_t) * pktcnt.
       * Otherwise, an error is returned.
       */
       if (write(fd, buf, len) < 0) {
           /* Error recovery. */
    }

       /*
       * Read length should be sum of all pkt descriptors
       * length + payload data length of all pkts
       * (sizeof(ugen_isoc_pkt_descr_t) + pktlen) * pktcnt
```

```
            */
            if (read(fd, rdbuf, (sizeof(ugen_isoc_pkt_descr_t) +
             pktlen) * pktcnt) < 0) {
                /* Error recovery. */
        }

            pktdesc = (ugen_isoc_pkt_descr_t *) rdbuf;

            /* points to payload beginning */
            p = rdbuf + pktcnt * sizeof(ugen_isoc_pkt_descr_t);

            for (i = 0; i < pktcnt; i++) {
                printf("packet %d len = %d,"
                    " actual_len = %d, status = 0x%x\n",
                    i, pktdesc->dsc_isoc_pkt_len,
                    pktdesc->dsc_isoc_pkt_actual_len,
                    pktdesc->dsc_isoc_pkt_status);

                /* Processing data */

                /*
                * next packet data payload, do NOT use
                * dsc_isoc_pkt_actual_len
                */
                p += pktdesc->dsc_isoc_pkt_len;

                pktdesc++;
}
```

For an isochronous-OUT endpoint, applications use the same packet descriptor and request structures to write request information to the ugen node. Following the packet request head information is the packet payload data. Upon successful return of write(2), applications can read(2) the same ugen file immediately to retrieve the individual packet transfer status of the last request. If the application isn't concerned about the status, it can omit it.

In the following example, an application transfers data on an isochronous-OUT endpoint:

```
#include <sys/usb/clients/ugen/usb_ugen.h>
char *buf, *p;
ushort_t i, pktlen;
int len, pktcnt;
ugen_isoc_req_head_t *req;
ugen_isoc_pkt_descr_t *pktdesc;
char rdbuf[4096];

pktcnt = 4;

/*
```

```
    * set packet length to a proper value, don't
    * exceed endpoint's capability
    */
    pktlen = 1024;

    len = sizeof(int) +
        sizeof(ugen_isoc_pkt_descr_t) * pktcount;

    len += pktlen * pktcnt;

    buf = malloc(len);
    if (!buf) {
        /* Error recovery. */
     }

    req = (ugen_isoc_req_head_t *)buf;
    req->req_isoc_pkts_count = pktcnt;

    pktdesc =
        (ugen_isoc_pkt_descr_t *)(req->req_isoc_pkt_descrs);

    for (i = 0; i < pktcnt; i++) {
        pktdesc[i].dsc_isoc_pkt_len = pktlen;
        pktdesc[i].dsc_isoc_pkt_actual_len = 0;
        pktdesc[i].dsc_isoc_pkt_status = 0;
     }

    /* moving to beginning of payload data */
    p = buf + sizeof(int) + sizeof(*pktdesc) * pktcnt;
    for (i = 0; i < pktcnt; i++) {

        /* fill in the data buffer */

        p += pktlen;
}

    /*
    * write packet request information and data to ugen driver
    *
    * len should be the exact value of sizeof(int) +
    *    sizeof(ugen_isoc_pkt_descr_t) * pktcnt + payload length
    */
    if (write(fd, buf, len) < 0) {
        /* Error recovery. */
     }

    /* read packet status */
```

```
if (read(fd, rdbuf, sizeof(*pktdesc) * pktcnt) < 0) {

    /* Error recovery. */

} else {

    /* Parse every packet's transfer status */

  }
```

Errors The following statuses are returned by endpoint status device logical names:

USB_LC_STAT_NOERROR	No error.
USB_LC_STAT_CRC	CRC error detected.
USB_LC_STAT_BITSTUFFING	Bit stuffing error.
USB_LC_STAT_DATA_TOGGLE_MM	Data toggle did not match.
USB_LC_STAT_STALL	Endpoint returned stall.
USB_LC_STAT_DEV_NOT_RESP	Device not responding.
USB_LC_STAT_UNEXP_PID	Unexpected Packet Identifier (PID).
USB_LC_STAT_PID_CHECKFAILURE	Check bits on PID failed.
USB_LC_STAT_DATA_OVERRUN	Data overrun.
USB_LC_STAT_DATA_UNDERRUN	Data underrun.
USB_LC_STAT_BUFFER_OVERRUN	Buffer overrun.
USB_LC_STAT_BUFFER_UNDERRUN	Buffer underrun.
USB_LC_STAT_TIMEOUT	Command timed out.
USB_LC_STAT_NOT_ACCESSED	Not accessed by the hardware.
USB_LC_STAT_UNSPECIFIED_ERR	Unspecified USBA or HCD error.
USB_LC_STAT_NO_BANDWIDTH	No bandwidth available.
USB_LC_STAT_HW_ERR	Host Controller h/w error.
USB_LC_STAT_SUSPENDED	Device was suspended.
USB_LC_STAT_DISCONNECTED	Device was disconnected.
USB_LC_STAT_INTR_BUF_FULL	Polling was stopped as the interrupt-IN data buffer was full. Buffer is now empty and polling has been resumed.
USB_LC_STAT_INTERRUPTED	Request was interrupted.

USB_LC_STAT_NO_RESOURCES	No resources available for request.
USB_LC_STAT_INTR_POLLING_FAILED	Failed to restart polling.
USB_LC_STAT_ISOC_POLLING_FAILED	Failed to start isochronous polling.
USB_LC_STAT_ISOC_UNINITIALIZED	Isochronous packet information not initialized.
USB_LC_STAT_ISOC_PKT_ERROR	All packets in this isochronous request have errors. The polling on this isochronous-IN endpoint is suspended and can be resumed on next `read(2)`.

The following system call `errno` values are returned:

`EINVAL`	An attempt was made to enable or disable "one transfer" mode while the associated endpoint was open.
`EBUSY`	The endpoint has been opened and another open is attempted.
`EACCES`	An endpoint open was attempted with incorrect flags.
`ENOTSUP`	Operation not supported.
`ENXIO`	Device associated with the file descriptor does not exist.
ENODEV	Device has been hot-removed or a suspend/resume happened before this command.
EIO	An I/O error occurred. Send a read on the endpoint status minor node to get the exact error information.
EINTR	Interrupted system call.
ENOMEM	No memory for the allocation of internal structures.

Files

```
/kernel/drv/ugen   32 bit ELF kernel module (x86 platform only)
/kernel/drv/sparcv9/ugen   64 bit ELF kernel module

/dev/usb/<vid>.<pid>/<N>/cntrl0
/dev/usb/<vid>.<pid>/<N>/cntrl0stat

/dev/usb/<vid>.<pid>/<N>/if<interface#>
                        <in|out|cntrl><endpoint#>
/dev/usb/<vid>.<pid>/<N>/if<interface#>
                        <in|out|cntrl><endpoint#>stat

/dev/usb/<vid>.<pid>/<N>/if<interface#>.
                        <alternate><in|out|cntrl<endpoint#>
/dev/usb/<vid>.<pid>/<N>/if<interface#>.
                        <alternate><in|out|cntrl><endpoint#>stat
```

```
/dev/usb/<vid>.<pid>/<N>/cfg<value>if<interface#>
                        <in|out|cntrl><endpoint#>
/dev/usb/<vid>.<pid>/<N>/cfg<value>if<interface#>
                        <in|out|cntrl<endpoint#stat>

/dev/usb/<vid>.<pid>/<N>/cfg<value>if<interface#>.
                        <alternate><in|out|cntrl><endpoint#>
/dev/usb/<vid>.<pid>/<N>/cfg<value>if<interface#>.
                        <alternate><in|out|cntrl><endpoint#>stat


/dev/usb/<vid>.<pid>/<N>/devstat

/dev/usb/<vid>.<pid>/<N>/if<interface#>cntrl0
/dev/usb/<vid>.<pid>/<N>/if<interface#>cntrl0stat
```

where *N* is an integer representing the instance number of this type of device. (All logical device names for a single device share the same *N*.)

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	PCI-based SPARC
Availability	SUNWugen, SUNWugenu

See Also `libusb(3LIB)`, `close(2)`, `poll(2)`, `read(2)`, `write(2)`, `aioread(3C)`, `aiowrite(3C)`, `usba(7D)`, `usb_dev_descr(9S)`.

Diagnostics In addition to being logged, the following messages may appear on the system console. All messages are formatted in the following manner:

```
Warning: <device path> (ugen<instance num>): Error Message...
```

Too many minor nodes.
: Device has too many minor nodes. Not all are available.

Instance number too high (<*number*>).
: Too many devices are using this driver.

Cannot access <device>. Please reconnect.
: This device has been disconnected because a device other than the original one has been inserted. The driver informs you of this fact by displaying the name of the original device.

Device is not identical to the previous one on this port. Please disconnect and reconnect.
: Same condition as described above; however in this case, the driver is unable to identify the original device with a name string.

Notes `ugen` returns `-1` for all commands and sets `errno` to `ENODEV` when device has been hot-removed or resumed from a suspend. The application must close and reopen all open minor nodes to reinstate successful communication.

Name uhci – host controller driver

Synopsis `pcivid,pid@unit-address`

Description The `uhci` host controller driver is a USBA (Solaris USB Architecture) compliant nexus driver that supports the *Universal Host Controller Interface Specification 1.1*, an industry standard developed by Intel. The `uhci` driver supports all USB transfers, including interrupt, control, isochronous and bulk.

Files

`/kernel/drv/uhci`	32–bit ELF kernel module. (SPARC or x86).
`/kernel/drv/amd64/uhci`	64–bit ELF kernel module. (x86).
`/kernel/drv/sparcv9/uhci`	64-bit SPARC ELF kernel module.
`/kernel/drv/uhci.conf`	Driver configuration file.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC or x86 PCI-based systems
Availability	SUNWusb

See Also `attributes(5)`, `ehci(7D)`, `hubd(7D)`, `ohci(7D)`, `usba(7D)`

Writing Device Drivers

Universal Host Controller Interface Specification for USB 1.1

Universal Serial Bus Specification 2.0

System Administration Guide: Basic Administration

http://*www.sun.com/io*

Diagnostics All host controller errors are passed to the client drivers. Root errors are documented in `hubd(7D)`.

In addition to being logged, the following messages may appear on the system console. All messages are formatted in the following manner:

```
WARNING: <device path> <uhci><instance number>>: Error message...
```

No SOF interrupts have been received. This USB UHCI host controller is unusable.
The USB hardware is not generating Start Of Frame interrupts. Please reboot the system. If this problem persists, contact your system vendor.

Name ural – Ralink RT2500USB 802.11b/g Wireless Driver

Description The `ural` *802.11b/g* wireless NIC driver is a multi-threaded, loadable, clonable, GLDv3-based STREAMS driver supporting the Ralink RT2500USB chipset-based NIC's.

Configuration The `ural` driver performs auto-negotiation to determine the data rate and mode. Supported 802.11b data rates are 1, 2, 5.5 and 11 Mbits/sec. Supported *802.11g* data rates are 1, 2, 5.5, 11, 6, 9, 12, 18, 24, 36, 48 and 54 Mbits/sec. The `ural` driver supports only BSS networks (also known as "ap" or "infrastructure" networks) and "open" (or "open-system") or "shared system" authentication.

Files `/dev/ural*`
Special character device.

`/kernel/drv/ural`
32-bit ELF kernel module. (x86)

`/kernel/drv/amd64/ural`
64-bit ELF kernel module. (x86)

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWural
Interface Stability	Committed

See Also `dladm(1M)`, `wificonfig(1M)`, `attributes(5)`, `gld(7D)`, `dlpi(7P)`

802.11 - Wireless LAN Media Access Control and Physical Layer Specification - IEEE, 2001

Name urtw – RealTek RTL8187L/RTL8187B USB 802.11b/g Wireless Driver

Description The `urtw` 802.11b/g wireless driver is a multi-threaded, loadable, clonable, GLDv3-basedSTREAMS driver supporting the RealTek RTL8187L chipset-based wireless devices.

Configuration The `urtw` driver performs auto-negotiation to determine the data rate and mode. Supported 802.11b data rates are 1, 2, 5.5 and 11 Mbits/sec. Supported 802.11g data rates are 1, 2, 5.5, 11, 6, 9, 12, 8, 24, 36, 48 and 54 Mbits/sec. The `atu` driver supports only BSS networks (also known `asap` or `infrastructure` networks) and `open` (or `open-system`) or `shared system` authentication.

Files

`/dev/urtw*`	Special character device.
`/kernel/drv/urtw`	32-bit ELF kernel module. (x86)
`/kernel/drv/amd64/urtw`	64-bit ELF kernel module. (x86)

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWurtw
Interface Stability	Committed

See Also `dladm(1M)`, `wificonfig(1M)`, `attributes(5)`, `gld(7D)`, `dlpi(7P)`

802.11 — Wireless LAN Media Access Control and Physical Layer Specification, IEEE, 2001

Name usba, usb – Solaris USB Architecture (USBA)

Description USB provides a low-cost means for attaching peripheral devices, including mass-storage devices, keyboards, mice, and printers, to a system. For complete information on the USB architecture, visit the USB website at http://www.usb.org.

USBA supports 126 hot-pluggable USB devices per USB bus. The maximum data transfer rate is 1.5 Mbits (low speed USB 1.x) or 12 Mbits (full speed USB 1.x) or 480 MBits (high speed USB 2.0) Mbits per second (Mbps).

USBA adheres to the *Universal Serial Bus 2.0* specification and provides a transport layer abstraction to USB client drivers.

For information on how to write USB client drivers, see *Writing Device Drivers*. For the latest information on writing USB drivers, visit *http://developers.sun.com/solaris/developer/support/driver/usb.html*. For a complete list of USBA interfaces, see `Intro(9F)` and `Intro(9S)`.

Devices without a driver may have a `libusb(3LIB)` application. For more information, see `/usr/sfw/share/doc/libusb/libusb.txt`.

Files Listed below are drivers and modules which either utilize or are utilized by USBA. Drivers in `/kernel/drv` are 32 bit drivers (x86 only). Drivers in `/kernel/drv/sparcv9` or `/kernel/drv/amd64` are 64 bit drivers.

Client Driver	*Function/Device*
`kernel/drv/[sparcv9|amd64/]hid`	`HID class`
`kernel/drv/[sparcv9|amd64/]hubd`	`hub class`
`kernel/drv/[sparcv9|amd64/]hwahc`	`HWA Host Controller class`
`kernel/drv/[sparcv9|amd64/]hwarc`	`HWA Radio Controller class`
`kernel/drv/[sparcv9|amd64/]scsa2usb`	`mass storage class`
`kernel/drv/[sparcv9|amd64/]usbprn`	`printer class`
`kernel/drv/[sparcv9|amd64/]usb_as`	`audio streaming class`
`kernel/drv/[sparcv9|amd64/]usb_ac`	`audio control class`
`kernel/drv/[sparcv9|amd64/]usbvc`	`video class`
`kernel/drv/[sparcv9|amd64/]usb_mid`	`multi-interface device`
`kernel/drv/[sparcv9|amd64/]usb_ia`	`interface-association driver`
`kernel/drv/[sparcv9|amd64/]usbser_edge`	`Edgeport USB to serial port`
`kernel/drv/[sparcv9|amd64/]usbsksp`	`Keyspan USB to serial port`
`kernel/drv/[sparcv9|amd64/]usbsprl`	`pl2303 USB to serial port`
`kernel/drv/[sparcv9|amd64/]usbsacm`	`CDC ACM class to serial port`
`kernel/drv/[sparcv9|amd64/]ugen`	`generic USB driver`
`kernel/drv/[sparcv9|amd64/]wusb_ca`	`WUSB Cable Association class`
`kernel/drv/[sparcv9|amd64/]ohci`	`open host controller driver`
`kernel/drv/[sparcv9|amd64/]uhci`	`universal host controller driver`
`kernel/drv/[sparcv9|amd64/]ehci`	`enhanced host controller driver`

Client Streams Modules *Function/Device*

```
/kernel/strmod/[sparcv9|amd64]usbkbm     Keyboad
/kernel/strmod/[sparcv9|amd64]usbms      Mouse
/kernel/strmod/[sparcv9|amd64]usb_ah     Audio HID
```

Host Controller Interface Drivers *Device*

```
/kernel/drv/[sparcv9|amd64]ehci     Enhanced HCI
/kernel/drv/[sparcv9|amd64]ohci     Open HCI
/kernel/drv/[sparcv|amd64/]uhci     Univeral HCI
```

Attributes See attributes(5) for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	PCI-based systems
Availability	SUNWusb, SUNWusbu

See Also cfgadm_usb(1M), libusb(3LIB), attributes(5), ehci(7D), hid(7D), hubd(7D), ohci(7D), scsa2usb(7D), uhci(7D), usb_ac(7D), usb_as(7D), usb_ia(7D), usb_mid(7D), usbprn(7D), usbsacm(7D), usbser_edge(7D), usbsksp(7D), usbsprl(7D), usbvc(7D), ugen(7D), virtualkm(7D). Intro(9F), Intro(9S)

Writing Device Drivers

Universal Serial Bus Specification 2.0.

Interface Association Descriptor Engineering Change Notice (ECN)

System Administration Guide: Basic Administration

http://www.sun.com

Notes Booting from USB mass-storage devices is not supported on SPARC, but is supported on X86.

Diagnostics The messages described below may appear on the system console as well as being logged. All messages are formatted in the following manner:

```
WARNING: Error message...
```

No driver found for device <device_name> (interface <number> node name=<node_name>)
The installed Solaris software does not contain a supported driver for this hardware. <number> is the interface number. <name> is either the device path name or the device name.

Draining callbacks timed out!
An internal error occured. Please reboot your system. If this problem persists, contact your system vendor.

The following messages may be logged into the system log. They are formatted in the following manner:

```
<device path><usba<instance number>): message...
```

Incorrect USB driver version for <n.m>. Driver is incompatible with USBA framework.

Name usb_ac – USB audio control driver

Synopsis `sound-control@unit-address`

Description The `usb_ac` driver is a USBA (Solaris USB Architecture) compliant client driver that supports the *USB Audio Class 1.0* specification.

The audio control driver is a USB class driver and offers functionality similar to the `audiocs` (sun4u) and `audiots` (Sun Blade 100) drivers which use the Solaris audio mixer framework (`mixer(7I)`). Unlike the `audiocs` and `audiots` drivers, the USB audio device can have play-only or record-only capability.

Drivers corresponding to other USB audio interfaces on the device, including the `usb_as(7D)` audio streaming driver or the `hid(7D)` driver, are plumbed under the USB audio control driver and do not directly interface with user applications.

The `usb_ac` driver supports USB audio class compliant devices with a feature unit.

Errors If a device is hot-removed while it is active, all subsequent opens returns EIO. All other errors are defined in the `audio(7I)` man page.

Files

`/kernel/drv/usb_ac`	32-bit x86 ELF kernel module
`/kernel/drv/amd64/usb_ac`	64-bit x86 ELF kernel module
`/kernel/drv/sparcv9/usb_ac`	64-bit SPARC ELF kernel module.
`/kernel/drv/usb_ac.conf`	USB audio driver configuration file.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86, PCI-based systems
Availability	SUNWusb
Interface Stability	Uncommitted

See Also `cfgadm_usb(1M)`, `ioctl(2)`, `attributes(5)`, `hid(7D)`, `usba(7D)`, `usb_as(7D)`, `audio(7I)`, `mixer(7I)`, `streamio(7I)`, `usb_ah(7M)`

Writing Device Drivers

Universal Serial Bus Specification 1.0 and 1.1

Universal Serial Bus Device Class Definition for Audio Devices, Release 1.0

System Administration: Basic Administration

Diagnostics In addition to being logged, the following messages can appear on the system console. All messages are formatted in the following manner:

```
Warning: <device path> (usb_ac<instance num>): Error Message...
```

Failure to plumb audio streams drivers.	The `usb` audio streaming driver or the `hid` driver could not be plumbed under the audio control driver and the device is not usable.

Name usb_ah – USB audio HID STREAMS module

Description The `usb_ah` STREAMS module enables the USB input control device which is a member of the Human Interface Device (HID) class and provides support for volume change and mute button. The `usb_ah` module is pushed on top of a HID class driver instance (see `hid(7D)`) and below an Audio Control class driver instance (see `usb_ac(7D)`). It translates the HID specific events to the events that are supported by the Solaris audio mixer framework.

Files

`/kernel/strmod/usb_ah`	32-bit ELF kernel STREAMS module. (x86 platform only.)
`/kernel/strmod/sparcv9/usb_ah`	SPARC 64-bit ELF kernel STREAMS module
`/kernel/strmod/amd64/usb_ah`	x8664-bit ELF kernel STREAMS module

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	PCI-based systems
Availability	SUNWusb
Interface Stability	Committed

See Also `hid(7D)`, `usba(7D)`, `usb_ac(7D)`, `usb_as(7D)`, `usb_mid(7D)`, `audio(7I)`,

STREAMS Programming Guide

System Administration Guide: Basic Administration

Universal Serial Bus Specification 1.0 and 1.1

Device Class Definition for Human Interface Devices (HID) 1.1

Diagnostics None

Notes If USB audio drivers are not loaded, buttons are not active.

Name usb_as – USB audio streaming driver

Synopsis `sound@unit-address`

Description The `usb_as` driver is a USBA (Solaris USB Architecture) compliant client driver that supports the *USB Audio Class 1.0* specification.

The `usb_as` driver processes audio data messages during play and record and sets sample frequency, precision, encoding and other functions on request from the USB audio control driver. See `usb_ac(7D)`.

This driver is plumbed under the USB audio control driver and does not directly interface with the user application.

Files

`/kernel/drv/usb_as`	32–bit x86 ELF kernel module
`/kernel/drv/amd64/usb_as`	64–bit x86 ELF kernel module
`/kernel/drv/sparcv9/usb_as`	64–bit SPARC ELF kernel module

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86, PCI-based systems
Availability	SUNWusb
Stability level	Committed

See Also `attributes(5)`, `usba(7D)`, `usb_ac(7D)`, `audio(7I)`

Writing Device Drivers

Universal Serial Bus Specification 1.0 and 1.1

System Administration Guide: Basic Administration

Diagnostics In addition to being logged, the following messages can appear on the system console. All messages are formatted in the following manner:

```
Warning: <device path> (usb_as<instance num>): Error Message...
```

where <*device path*> is the physical path to the device in `/devices` directory.

No bandwidth available.
: There is no bandwidth available for the isochronous pipe. As a result, no data is transferred during play and record.

Operating a full/high speed audio device on a high speed port is not supported.
The USB software does not currently support full or high speed audio devices connected to an external USB 2.0 hub that is linked to a port of a USB 2.0 host controller. Audio devices must be connected directly to a port of a USB 2.0 controller or to any USB 1.1 port.

Cannot access device. Please reconnect <name>.
There was an error in accessing the device during reconnect. Please reconnect the device.

Device is not identical to the previous one on this port. Please disconnect and reconnect.
A USB audio streaming interface was hot-removed while open. A new device was hot-inserted which is not identical to the original USB audio device. Please disconnect the USB device and reconnect the device to the same port.

Notes The USB audio streaming interface is power managed if the device is idle.

Name usbftdi – FTDI USB to serial converter driver

Synopsis

```
#include <fcntl.h>
#include <sys/termio.h>
usbftdi@unit
```

Description The `usbftdi` driver is a loadable STREAMS and USBA (Solaris USB Architecture) compliant client driver that provides basic asynchronous communication support for FTDI USB-to-serial converters. Serial device streams are built with appropriate modules that are pushed atop the `usbftdi` driver by the `autopush(1M)` facility.

Application Programming Interface

The `usbftdi` module supports the `termio(7I)` device control functions specified by flags in the `c_cflag` word of the `termios` structure, and by the `IGNBRK, IGNPAR, PARMRK`, and `INPCK` flags in the `c_iflag` word of the `termios` structure. All other `termio(7I)` functions must be performed by STREAMS modules pushed atop the driver. When a device is opened, the , `ldterm(7M)` and `ttcompat(7M)` STREAMS modules are automatically pushed on top of the stream, providing the standard `termio(7I)` interface.

Use device logical names `/dev/term/[0-9]*` to access the serial ports for a dial-in line that is used with a modem.

Use device logical names `/dev/cua/[0-9]*` to access the serial ports for other applications. These names are also used to provide a logical access point for a dial-out line.

Device hot-removal is functionally equivalent to a modem disconnect event, as defined in `termio(7I)`.

Input and output line speeds can be set to the following baud rates: 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800 or 921600. Input and output line speeds can not be set independently. For example, when the output speed is set, the input speed is automatically set to the same speed.

Soft Carrier Capabilities

Many devices that use this USB serial interface component are not, in fact dial-in lines connected to carefully configured RS-232 modems. They are often intelligent peripherals whose manufacturers want to present a serial port interface to application software. Some applications use only three wire connections, or are otherwise somewhat casual about the state of the Carrier Detect (electrical) signal, and the other modem control lines.

The configuration file delivered with this driver, `usbftdi.conf`, acknowledges this by setting the driver property `ignore-cd` to 1. This enables `soft carrier` mode where the kernel does *not* block opens waiting for DCD to be asserted.

This behavior also matches the default `ignore carrier detect` behavior of the onboard serial ports of machines that have them. See `eeprom(1M)` for further details.

The `hardware carrier` behavior (the driver's internal default) can be selected by either unsetting (commenting out) the `ignore-cd` property, or by setting the value of the property to zero.

More sophisticated selection of which devicesl ignore or obey the DCD signal can be effected using `port-%d-ignore-cd` properties.

Dial-In and Dial-Out Support

A related feature is available for traditional usage that enables a single tty line to be connected to a modem and used for incoming and outgoing calls. By accessing through device logical name `/dev/cua/[0-9]*`, you can open a port without the carrier detect signal being asserted, either through hardware or an equivalent software mechanism. These devices are commonly known as dial-out lines.

A dial-in line can be opened only if the corresponding dial-out line is closed. A blocking `/dev/term` open waits until the `/dev/cua` line is closed, which drops Data Terminal Ready, after which Carrier Detect usually drops as well. When the carrier is detected again with the `/dev/cua` device remaining closed, this indicates an incoming call and the blocking open seizes exclusive use of the line.

A non-blocking `/dev/term` open returns an error if the `/dev/cua` device is open.

If the `/dev/term` line is opened successfully (usually only when carrier is recognized on the modem, though see `Soft Carrier Capabilities` section of this manual page), the corresponding `/dev/cua` line can not be opened. This allows a modem and port to be used for dial-in (enabling the line for login in `/etc/inittab`) or dial-out (using `tip(1)` or `uucp(1C)`) when no-one is logged in on the line.

Errors

An `open()` fails under the following conditions:

`ENXIO` The unit being opened does not exist.

`EBUSY` The `/dev/cua` (dial-out) device is being opened while the `/dev/term` (dial-in device) is open, or the dial-in device is being opened with a no-delay open while the dial-out device is open.

`EBUSY` The unit has been marked as exclusive-use by another process with a `TIOCEXCL ioctl()` call.

`EIO` USB device I/O error.

Files

`/usr/kernel/drv/usbftdi`	32-bit x86 ELF kernel module
`/usr/kernel/drv/usbftdi.conf`	Kernel module configuration file
`/usr/kernel/drv/amd64/usbftdi`	64-bit x86 ELF kernel module
`/usr/kernel/drv/sparcv9/usbftdi`	64-bit SPARC ELF kernel module
`/dev/cua/[0-9]*`	Dial-out tty lines
`/dev/term/[0-9]*`	Dial-in tty lines

Attributes See attributes(5) for a description of the following attribute:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86, PCI-based systems
Availability	SUNWuftdi

See Also strconf(1), tip(1), uucp(1C), autopush(1M), eeprom(1M), ioctl(2), open(2), termios(3C), usba(7D), termio(7I), ldterm(7M), ttcompat(7M), eeprom(1M), attributes(5),

Diagnostics In addition to being logged, the following messages might appear on the system console. All messages are formatted in the following manner:

`Warning:` *device_path* `usbftdi`*instance num*`): Error Message ...`

`Device was disconnected while open. Data may have been lost.`
The device has been hot-removed or powered off while it was open and a possible data transfer was in progress. The job might be aborted.

`Device is not identical to the previous one on this port. Please disconnect and reconnect.`
The device was hot-removed while open. A new device was hot-inserted which is not identical to the original device. Please disconnect the device and reconnect the original device to the same port.

`Device has been reconnected, but data may have been lost.`
The device that was hot-removed from its USB port has been re-inserted again to the same port. It is available for access but data from a previous transfer might be lost.

`Cannot access` *device*`. Please reconnect.`
This device has been disconnected because a device other than the original one has been inserted. The driver informs you of this fact by displaying the name of the original device.

The following messages might be logged into the system log. They are formatted in the following manner:

device_path `usbftdi`*instance number*`): message ...`

`Input overrun.` Data was lost.

Name usb_ia – USB interface association driver

Synopsis

```
interface association@unit-address
```

Description The `usb_ia` driver is a USBA (Solaris Universal Serial Bus Architecture)-compliant nexus driver that binds to a device's interface association nodes when no vendor or class specific driver is available. To do this, `usb_ia` creates nodes for the internal interfaces and then attempts to bind drivers to each child interface.

Each interface association node has a parent device node that is created by `usb_mid(7D)` and all `ugen(7D)` interfaces are exported by `usb_mid(7D)`. (Note: attempting to export `ugen(7D)` interfaces using `usb_ia` is prohibited.)

Files

`/kernel/drv/usb_ia`
32-bit ELF kernel module. (x86).

`/kernel/drv/amd64/usb_ia`
64-bit ELF kernel module. (x86).

`/kernel/drv/sparcv9/usb_ia`
64-bit ELF kernel module. (SPARC).

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC & x86 PCI-based systems
Availability	SUNWusb

See Also `attributes(5)`,`ugen(7D)`, `usb_mid(7D)`, `usba(7D)`

Writing Device Drivers

System Administration Guide: Basic Administration

Universal Serial Bus Specification 2.0 — 2000

Interface Association Descriptor Engineering Change Notice (ECN)—2003

http://www.sun.com/io

http://docs.sun.com

Diagnostics In addition to being logged, the following messages may appear on the system console. All messages are formatted in the following manner:

```
<device path> (usb_ia<instance num>):message...
```

No driver found for interface <n> (nodename: <string>) of <device>.
No driver is available for this interface.

Name usbkbm – keyboard STREAMS module for Sun USB Keyboard

Synopsis

```
open("/dev/kbd", O_RDWR)
```

Description The `usbkbm` STREAMS module processes byte streams generated by a keyboard attached to a USB port. USB keyboard is a member of Human Interface Device (HID) Class, and `usbkbm` only supports the keyboard protocol defined in the specification. Definitions for altering keyboard translation and reading events from the keyboard are in `<sys/kbio.h>` and `<sys/kbd.h>`.

The `usbkbm` STREAMS module adheres to the interfaces exported by kb(7M). Refer to the `DESCRIPTION` section of kb(7M) for a discussion of the keyboard translation modes and the `IOCTL` section for the supported `ioctl(2)` requests.

IOCTLS USB Keyboard `usbkbm` returns different values for the following ioctls than kb(7M):

`KIOCTYPE` This `ioctl()` returns a new keyboard type defined for the USB keyboard. All types are listed below:

```
KB_SUN3   Sun Type 3 keyboard
KB_SUN4   Sun Type 4 keyboard
KB_ASCII  ASCII terminal masquerading as keyboard
KB_PC     Type 101 PC keyboard
KB_USB    USB keyboard
```

The USB keyboard type is `KB_USB`; `usbkbm` will return `KB_USB` in response to the `KIOCTYPE` ioctl.

`KIOCLAYOUT` The argument is a pointer to an `int`. The layout code specified by the `bCountryCode` value returned in the HID descriptor is returned in the int pointed to by the argument. The `countrycodes` are defined in 6.2.1 of the HID 1.0 specifications.

`KIOCCMD`

`KBD_CMD_CLICK/KBD_CMD_NOCLICK`	The kb(7M) indicates that inappropriate commands for particular keyboards are ignored. Because clicking is not supported on the USB keyboard, `usbkbm` ignores this command
`KBD_CMD_SETLED`	Set keyboard LEDs. Same as kb(7M).
`KBD_CMD_GETLAYOUT`	The country codes defined in 6.2.1 of the HID 1.0 specification are returned.

KBD_CMD_BELL/KBD_CMD_NOBELL	This command is supported although the USB keyboard does not have a buzzer. The request for the bell is rerouted.
KBD_CMD_RESET	There is no notion of resetting the keyboard as there is for the type4 keyboard. usbkbm ignores this command and does not return an error.

Files

/kernel/strmod/usbkbm	32-bit ELF kernel STREAMS module (x86 platform only)
/kernel/strmod/sparcv9/usbkbm	SPARC 64-bit ELF kernel STREAMS module

Attributes See attributes(5) for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	PCI-based systems
Availability	SUNWusb

See Also dumpkeys(1), kbd(1), loadkeys(1), ioctl(2), keytables(4), attributes(5), hid(7D), usba(7D), virtualkm(7D), kb(7M)

STREAMS Programming Guide

System Administration Guide: Basic Administration

http://*www.sun.com/io*

Diagn0STICS None

Name usb_mid – USB Multi Interface Driver

Synopsis `device@`*unit-address*

Description The `usb_mid` driver is a USBA (Solaris Universal Serial Bus Architecture)-compliant nexus driver that binds to device level nodes of a composite (multi interface) device if no vendor or class-specific driver is available. The `usb_mid` driver creates interface nodes or interface association nodes and attempts to bind drivers to them. If no driver is found for interface association nodes, `usb_ia(7D)` is bound by default.

UGEN (Generic USB) The `usb_mid` driver also supports a `ugen(7D)` interface allowing raw access to the device, for example by `libusb(3LIB)` applications, by passing the drivers bound to each interface. Since a libusb application might change the state of the device, you should not access the device through the child interface drivers. Note that the `usb_mid` driver creates a ugen interface only if none of its children are explicitly bound to the `ugen(7D)` driver. Additionally, `usb_mid` does not create children.

Files

`/kernel/drv/usb_mid`	32-bit x86 ELF kernel module
`/kernel/drv/amd64/usb_mid`	64-bit x86 ELF kernel module
`/kernel/drv/sparcv9/usb_mid`	64-bit SPARC ELF kernel module
/dev/usb/*/*/*	ugen(7D) nodes.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC & x86, PCI-based systems
Availability	SUNWusb

See Also `cfgadm_usb(1M)`, `libusb(3LIB)`, `attributes(5)`, `usba(7D)`, `usb_ia(7D)`

Writing Device Drivers

Universal Serial Bus Specification 2.0—2000

Interface Association Descriptor Engineering Change Notice (ECN)—2003

System Administration Guide: Basic Administration

http://*www.sun.com/io*

Diagnostics In addition to being logged, the following messages may appear on the system console. All messages are formatted in the following manner:

```
Warning: <device path> (usb_mid<instance number>): Error Message...
```

Cannot access <device>. Please reconnect.
This device has been disconnected because a device other than the original one has been inserted. The driver informs you of this fact by displaying the name of the original device.

Device not identical to the previous one on this port. Please disconnect and reconnect.
Same condition as described above; however in this case, the driver is unable to identify the original device with a name string.

Please disconnect and reconnect this device.
A hotplug of the device is needed before it can be restored.

The following messages may be logged into the system log. They are formatted in the following manner:

```
<device path><usb_mid<instance number>): message...
```

No driver found for interface <n> (nodename: <string>) of <device>.
No driver is available for this interface.

No driver found for device <device>.
No driver is available for this interface.

Can't support ugen for multiple configuration devices that have attached child interface drivers.
No ugen interface is available and `libusb(3LIB)` cannot work with this device.

Name usbms – USB mouse STREAMS module

Synopsis

```
#include <sys/vuid_event.h>

#include <sys/vuid_wheel.h>

#include <sys/msio.h>

#include <sys/msreg.h>
```

Description The `usbms` STREAMS module processes byte streams generated by a USB mouse. A USB mouse is a member of the Human Interface Device (HID) class and the `usbms` module supports only the mouse boot protocol defined in the HID specification.

The `usbms` module must be pushed on top of the HID class driver (see `hid(7D)`). In the `VUID_FIRM_EVENT` mode, the `usbms` module translates packets from the USB mouse into Firm events. The Firm event structure is defined in `<sys/vuid_event.h>`. The STREAMS module state is initially set to raw or `VUID_NATIVE` mode which performs no message processing. See the *HID 1.0* specification for the raw format of the mouse packets. To initiate mouse protocol conversion to Firm events, change the state to `VUID_FIRM_EVENT`.

When the usb mouse is opened or hot plugged in, the MOUSE_TYPE_ABSOLUTE event (Firm event) is sent to the upper level to notify the VUID application that it is the absolute mouse.

ioctls

`VUIDGFORMAT` This option returns the current state of the STREAMS module. The state of the `usbms` STREAMS module may be either `VUID_NATIVE` (no message processing) or `VUID_FIRM_EVENT` (convert to Firm events).

`VUIDSFORMAT` The argument is a pointer to an `int`. Set the state of the STREAMS module to the `int` pointed to by the argument.

```
typedef struct  vuid_addr_probe {
     short base; /* default vuid device addr directed too */
     union {
            short next;   /* next addr for default when VUIDSADDR */
            short current; /* current addr of default when VUIDGADDR */
     } data;
} Vuid_addr_probe;
```

`VUIDSADDR` The argument is a pointer to a `Vuid_addr_probe` structure. `VUIDSADDR` sets the virtual input device segment address indicated by base to next.

If base does not equal `VKEY_FIRST`, `ENODEV` is returned.

`VUIDGADDR` The argument is a pointer to a `Vuid_addr_probe` structure. Return the address of the virtual input device segment indicated by base to current.

If base does not equal `VKEY_FIRST`, `ENODEV` is returned.

VUIDGWHEELCOUNT

This ioctl takes a pointer to an integer as argument and sets the value of the integer to the number of wheels available on this device. This ioctl returns 1 if wheel(s) are present and zero if no wheels are present.

VUIDGWHEELINFO

This command returns static information about the wheel that does not change while a device is in use. Currently the only information defined is the wheel orientation which is either VUID_WHEEL_FORMAT_VERTICAL or VUID_WHEEL_FORMAT_HORIZONTAL. If the module cannot distinguish the orientation of the wheel or the wheel is of some other format, the format is set to VUID_WHEEL_FORMAT_UNKNOWN.

```
typedef struct {
        int     vers;
        int     id;
        int     format;
} wheel_info;
```

The ioctl takes a pointer to "wheel_info" structure with the "vers" set to the current version of the "wheel_info" structure and "id" set to the id of the wheel for which the information is desired.

VUIDSWHEELSTATE/VUIDGWHEELSTATE

VUIDSWHEELSTATE sets the state of the wheel to that specified in the stateflags. VUIDGWHEELSTATE returns the current state settings in the stateflags field.

stateflags is an OR'ed set of flag bits. The only flag currently defined is VUID_WHEEL_STATE_ENABLED.

When stateflags is set to VUID_WHEEL_STATE_ENABLED the

module converts motion of the specified wheel into VUID events and sends those up stream.

Wheel events are enabled by default.

Applications that want to change the stateflags should first get the current stateflags and then change only the bit they want.

```
typedef struct {
        int             vers;
        int             id;
        uint32_t        stateflags;
} wheel_state;
```

These ioctls take a pointer to "wheel_state" as an argument with the "vers" and "id" members filled in. These members have the same meaning as that for 'VUIDGWHEEL INFO' ioctl.

`ioctl()` requests for changing and retrieving mouse parameters use the `Ms_parms` structure:

```
typedef struct {
     int     jitter_thresh;
     int     speed_law;
     int     speed_limit;
} Ms_parms;
```

`jitter_thresh` is the "jitter threshold" of the mouse. Motions fewer than `jitter_thresh` units along both axes are accumulated and then sent up the stream after 1/12 second.

`speed_law` indicates whether extremely large motions are to be ignored. If it is `1`, a "speed limit" is applied to mouse motions. Motions along either axis of more than `speed_limit` units are discarded.

`MSIOGETPARMS` The argument is a pointer to a `Ms_params` structure. The `usbms` module parameters are returned in the structure.

`MSIOSETPARMS` The argument is a pointer to a `Ms_params` structure. The `usbms` module parameters are set according to the values in the structure.

`MSIOSRESOLUTION` Used by the absolute mouse to get the current screen resolution. The parameter is a pointer to the `Ms_screen_resolution` structure:

```
int    height;         /* height of the screen */
int    width;         /* width of the screen */
}Ms_screen_resolution;
```

The `usbms` module parameters are set according to the values in the structure and used to calculate the correct coordinates.

Files

/kernel/strmod/usbms	32-bit ELF kernel STREAMS module (x86 platform only.)
/kernel/strmod/sparcv9/usbms	SPARC 64-bit ELF kernel STREAMS module

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	PCI-based systems
Availability	SUNWusb

See Also `ioctl(2)`, `attributes(5)`, `hid(7D)`, `virtualkm(7D)`, `usba(7D)`

System Administration Guide: Basic Administration

http://*www/sun.com/io*

Diagnostics The following messages may be logged into the system log. They are formatted in the following manner:

```
<device path><usbms<instance number>): message...
```

Invalid Hid descriptor tree. Set to default value (3 buttons).
The mouse supplied incorrect information in its HID report.

Mouse buffer flushed when overrun.
Mouse data was lost.

Name usbprn – USB printer class driver

Synopsis

```
#include <sys/usb/clients/printer/usb_printer.h>

#include <sys/ecppio.h>

usbprn@unit-address
```

Description The `usbprn` driver is a USBA (Solaris USB Architecture) compliant client driver that supports the *USB Printer Class 1.0* specification. The `usbprn` driver supports a subset of the `ecpp`(7D) parallel port driver functionality. However, unlike the STREAMS-based `ecpp` driver, `usbprn` is a character driver.

The `usbprn` driver supports all USB printer-class compliant printers. For a list of recommended printers and USB parallel printer adapters, visit http://*www.sun.com/io*.

The `usbrpn` driver includes support for communicating with many different printers. To use these printers, it may be nesessary to install and configure additional format conversion packages available in the Solaris distribution. Configuration of these conversion packages under the Solaris printing system can be simplified through the use of the `printmgr`(1M). This tool allows selection of printer manufacturer/model information while creating a print queue. For USB connected printers, it attempts to pre-select the the manufacturer and model information based on the 1284 device id supplied by the printer.

UGEN (Generic USB) The `usbprn` driver also supports a `ugen`(7D) interface allowing raw access to the device, for example by `libusb`(3LIB) applications, by passing the drivers bound to each interface. Because a libusb application might change the state of the device, you should not access the device through the child interface drivers.

Default Operation With certain minor exceptions (outlined in the Notes sections below), the `usbprn` driver supports a subset of the `ecpp`(7D) ioctl interfaces:

Configuration variables are set to their default values each time the USB printer device is attached. The `write_timeout` period (defined in the ECPPIOC_SETPARMS ioctl description below) is set to 90 seconds. The mode is set to centronics mode (ECPP_CENTRONICS). Parameters can be changed through the ECPPIOC_SETPARMS ioctl and read through the ECPPIOC_GETPARMS ioctl. Each time the USB printer device is opened, the device is marked as busy and all further opens will return EBUSY. Once the device is open, applications can write to the device and the driver can send data and obtain device id and status.

Note – Unlike the `ecpp`(7D) driver, `usbprn` resets configuration variables to their default values with each `attach`(9E). (The `ecpp`(7D) driver resets configuration variables with each `open`(2).)

Write Operation A `write`(2) operation returns the number of bytes successfully written to the device. If a failure occurs while a driver is transferring data to printer, the contents of the status bits are captured at the time of the error and can be retrieved by the application program using the ECPPIOC_GETERR `ioctl`(2) call. The captured status information is overwritten each time an ECPPIOC_TESTIO `ioctl`(2) occurs.

ioctls The usbprn driver supports prnio(7I) interfaces. Note that the PRNIOC_RESET command has no effect on USB printers.

The following ioctl(2) calls are supported for backward compatibility and are not recommended for new applications.

ECPPIOC_GETPARMS — Gets current transfer parameters. The argument is a pointer to struct ecpp_transfer_parms. If parameters are not configured after the device is opened, the structure will be set to its default configuration.

Note – Unlike the ecpp(7D) driver, only the ECPP_CENTRONICS mode is currently supported in usbprn.

ECPPIOC_SETPARMS — Sets transfer parameters. The argument is a pointer to a struct ecpp_transfer_parms. If a parameter is out of range, EINVAL is returned. If the peripheral or host device cannot support the requested mode, EPROTONOSUPPORT is returned.

The transfer parameters structure is defined in <sys/ecppio.h>:

```
struct ecpp_transfer_parms {
    int  write_timeout;
    int  mode;
};
```

The write_timeout field, which specifies how long the driver will take to transfer 8192 bytes of data to the device, is set to a default value of 90 seconds. The write_timeout field must be greater than one second and less than 300 seconds (five minutes.)

Note – Unlike the ecpp(7D) driver, only the ECPP_CENTRONICS mode is currently supported in usbprn. Also, the semantics of write_timeout in usbprn differ from ecpp(7D). Refer to ecpp(7D) for information.

BPPIOC_TESTIO — Tests the transfer readiness of a print device and checks status bits to determine if a write(2) will succeed. If status bits are set, a transfer will fail. If a transfer will succeed, zero is returned. If a transfer fails, the driver returns EIO and the state of the status bits are captured. The captured status can be retrieved using the BPPIOC_GETERR ioctl(2) call. BPPIOC_TESTIO and BPPIOC_GETERR are compatible to the ioctls specified in bpp(7D).

Note – Unlike the ecpp(7D) driver, only the ECPP_CENTRONICS mode is currently supported in usbprn. Additionally, bus_error and timeout_occurred fields are not used in the usbprn interface. (In ecpp(7D), timeout_occurred is used.)

`BPPIOC_GETERR` Get last error status. The argument is a pointer to a `struct bpp_error_status`. This structure indicates the status of all the appropriate status bits at the time of the most recent error condition during a `write(2)` call, or the status of the bits at the most recent BPPIOC_TESTIO `ioctl(2)` call.

```
struct bpp_error_status {
   char    timeout_occurred; /* not used */
   char    bus_error;        /* not used */
   uchar_t pin_status;       /* status of pins which
                             /* could cause error */
};
```

The pin_status field indicates possible error conditions. The error status structure `bpp_error_status` is defined in the include file `<sys/bpp_io.h>`. The valid bits for `pin_status` can be `BPP_ERR_ERR`, `BPP_SLCT_ERR`, and `BPP_PE_ERR`. A set bit indicates that the associated pin is asserted.

Note – Unlike the `ecpp(7D)` driver, only the ECPP_CENTRONICS mode is currently supported in `usbprn`. Additionally, the `bus_error` and `timeout_occurred` fields are not used in the `usbprn` interface. (In `ecpp(7D)`, `timeout_occurred` is used.) Unlike `ecpp(7D)`, the BPP_BUSY_ERR status bit is not supported by USB printers.

`ECPPIOC_GETDEVID` Gets the IEEE 1284 device ID from the peripheral. The argument is a pointer to a `struct ecpp_device_id`. Applications should set mode to ECPP_CENTRONICS. If another mode is used, the driver will return `EPROTONOSUPPORT`. `len` is the length of the buffer pointed to by `addr`. `rlen` is the actual length of the device ID string returned from the peripheral. If the returned `rlen` is greater than `len`, the application should call ECPPIOC_GETDEVID a second time with a buffer length equal to `rlen`.

The 1284 device ID structure:

```
struct ecpp_device_id {
  int  mode; /* mode to use for reading device id */
  int  len;  /* length of buffer */
  int  rlen;  /* actual length of device id string */
  char *addr; /* buffer address */
```

Note – Unlike `ecpp(7D)`, only the ECPP_CENTRONICS mode is currently supported in `usbprn`.

Read Operation The `read` operation is not supported and returns `EIO`.

Errors

`EBUSY`	The device has been opened and another open is attempted. An attempt has been made to unload the driver while one of the units is open.
`EINVAL`	An unsupported IOCTL has been received. A ECPPIOC_SETPARMS `ioctl(2)` is attempted with an out of range value in the `ecpp_transfer_parms` structure.
`EIO`	The driver has received an unrecoverable device error, or the device is not responding, or the device has stalled when attempting an access. A `write(2)` or `ioctl(2)` did not complete due to a peripheral access. A `read(2)` system call has been issued.
`ENXIO`	The driver has received an `open(2)` request for a unit for which the attach failed.
`ENODEV`	The driver has received an `open(2)` request for a device that has been disconnected.
`EPROTONOSUPPORT`	The driver has received a ECPPIOC_SETPARMS `ioctl(2)` for a mode argument other than ECPP_CENTRONICS in the `ecpp_transfer_parms` structure.

Files

`/kernel/drv/usbprn`	32–bit x86 ELF kernel module
`/kernel/drv/amd64/usbprn`	64–bit x86 ELF kernel module
`/kernel/drv/sparcv9/usbprn`	64–bit SPARC ELF kernel module
/dev/usb/*/*/*	ugen(7D) nodes.
`/dev/printers/`*n*	Character special files

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86, PCI-based systems
Availability	SUNWusb

See Also

`cfgadm_usb(1M)`, `printmgr(1M)`, `ioctl(2)`, `open(2)`, `read(2)`, `write(2)`, `libusb(3LIB)`, `attributes(5)`, `bpp(7D)`, `ecpp(7D)`, `ugen(7D)`, `usba(7D)prnio(7I)`, `attach(9E)`

Writing Device Drivers

Universal Serial Bus Specification 1.0 and 1.1

USB Device Class Definition for Printing Devices 1.0

System Administration Guide: Basic Administration

http://*www.sun.com/io*

Diagnostics In addition to being logged, the following messages may appear on the system console. All messages are formatted in the following manner:

```
Warning: <device path> (usbprn<instance num>): Error Message...
```

Device was disconnected while open. Data may have been lost.
The device has been hot-removed or powered off while it was open and a possible data transfer was in progress. The job may be aborted.

Cannot access <device>. Please reconnect.
There was an error in accessing the printer during reconnect. Please reconnect the device.

Device is not identical to the previous one on this port. Please disconnect and reconnect.
A USB printer was hot-removed while open. A new device was hot-inserted which is not identical to the original USB printer. Please disconnect the USB device and reconnect the printer to the same port.

Printer has been reconnected but data may have been lost.
The printer that was hot-removed from its USB port has been re-inserted again to the same port. It is available for access but the job that was running prior to the hot-removal may be lost.

Notes The USB printer will be power managed if the device is closed.

If a printer is hot-removed before a job completes, the job is terminated and the driver will return EIO. All subsequent opens will return `ENODEV`. If a printer is hot-removed, an LP reconfiguration may not be needed if a printer is re-inserted on the same port. If re-inserted on a different port, an LP reconfiguration may be required.

The USB Parallel Printer Adapter is not hotpluggable. The printer should be connected to USB Parallel Printer Adapter before plugging the USB cable into host or hub port and should be removed only after disconnecting the USB cable of USB Parallel Printer Adapter from the host or hub port.

Name usbsacm – USB communication device class ACM driver

Synopsis

```
#include <sys/termio.h>

usbsacm@unit
```

Description The `usbsacm` driver is a loadable STREAMS and USBA (Solaris USB architecture)-compliant client driver that provides basic asynchronous communication support for USB modems and ISDN terminal adapters that conform to the *Universal Serial Bus Communication Device Class Abstract Control Model (USB CDC ACM)* specification. You can download the *USB CDC* specification from the USB website at *http://www.usb.org/developers/devclass_docs/usbcdc11.pdf*. Supported devices include mobile phones and PCMCIA cards which provide modem function by the usb cable. Serial device streams are built with appropriate modules that are pushed atop the usbsacm driver by the `autopush(1M)` facility.

The `usbsacm` module supports the `termio(7I)` device control functions specified by flags in the c_cflag word of termios structure, and by the IGNBRK, IGNPAR, PARMRK and INPCK flags in the c_iflag word of the termios structure. All other `termio(7I)` functions must be performed by STREAMS modules pushed atop the driver. When a device is opened, the `ldterm(7M)` and `ttcompat(7M)` STREAMS modules are automatically pushed on top of the stream, providing the standard `termio(7I)` interface.

You use device logical names /dev/term/[*0-9*]* to access the serial ports. These names are typically used to provide a logical access point for a dial-in line that is used with a modem. You can use `pppd(1M)` to transmit datagrams over these serial ports.

A special feature (controlled by the minor device number) is available that enables a single tty line to be connected to a modem and used for incoming and outgoing calls. By accessing through device logical name /dev/cua/[*0-9*]*, you can open a port without the carrier detect signal being asserted, either through hardware or an equivalent software mechanism. These devices are commonly known as 'dial-out' lines.

Unlike onboard serial ports, the `usbsacm` ports cannot serve as a local serial console.

Application Programming Interface A dial-in line can be opened only if the corresponding dial-out line is closed. A blocking /dev/term open waits until the /dev/cua line is closed (which drops Data Terminal Ready, after which Carrier Detect usually drops as well) and carrier is detected again. A non-blocking /dev/term open returns an error if the /dev/cua is open.

If the /dev/term line is opened successfully (usually only when carrier is recognized on the modem), the corresponding /dev/cua line cannot be opened. This allows a modem and port to be used for dial-in (by enabling the line for login in `/etc/inittab`) or dial-out (by `tip(1)` or `uucp(1C)`) as `/dev/cua0` when no one is logged in on the line.

Device hot-removal is functionally equivalent to a modem disconnect event, as defined in `termio(7I)`.

ioctls The `usbsacm` driver supports the standard set of `termio(7I)` ioctl calls.

The input and output line speeds may be set to any of the following baud rates: 75, 150, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400 or 460800. The speeds cannot be set independently. For example, when the output speed is set, the input speed is automatically set to the same speed.

Errors An `open()` fails under the following conditions:

`ENXIO` The unit being opened does not exist.

`EBUSY` The /dev/cua (dial-out) device is being opened while the /dev/term (dial-in device) is open, or the dial-in device is being opened with a no-delay open while the dial-out device is open.

`EBUSY` The unit has been marked as exclusive-use by another process with a `TIOCEXCL` `ioctl()` call.

`EIO` USB device I/O error.

Files

`/kernel/drv/usbsacm`	32-bit ELF kernel module. (x86)
`/kernel/drv/amd64/usbsacm`	64-bit ELF kernel module. (x86)
`/kernel/drv/sparcv9/usbsacm`	64-bit ELF kernel module. (SPARC)
`/dev/cua/[0-9]`	dial-out tty lines
`/dev/term/[0-9]`	dial-in tty lines

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86 PCI-based systems
Availability	SUNWuacm

See Also `strconf(1)`, `tip(1)`,`uucp(1C)`, `autopush(1M)`, `pppd(1M)`, `ioctl(2)`, `open(2)`, `termios(3C)`, `attributes(5)`, `usba(7D)`, `termio(7I)`, `ldterm(7M)`, `ttcompat(7M)`

Diagnostics In addition to being logged, the following messages may appear on the system console. All messages are formatted in the following manner:

```
Warning: <device path> (usbsacm<instance num>):Error Message...
```

Device was disconnected while open. Data may have been lost.

The device has been hot-removed or powered off while it was open and a possible data transfer was in progress. The job may be aborted.

Device is not identical to the previous one on this port. Please disconnect and reconnect.
The device was hot-removed while open. A new device was hot-inserted which is not identical to the original device. Please disconnect the device and reconnect the original device to the same port.

Device has been reconnected, but data may have been lost.
The device that was hot-removed from its USB port has been re-inserted again to the same port. It is available for access but data from a previous transfer may be lost.

Cannot access <device>. Please reconnect.
This device has been disconnected because a device other than the original one has been inserted. The driver informs you of this fact by displaying the name of the original device.

The following messages may be logged into the system log. They are formatted in the following manner:

```
<device path><usbsacm<instance number>): message...
```

Input overrun. Data was lost.

Name usbser_edge – Digi Edgeport USB to serial converter driver

Synopsis

```
#include <fcntl.h>

#include <sys/termios.h>

usbser_edge@unit
```

Description The `usbser_edge` driver is a loadable STREAMS and USBA (Solaris USB architecture) compliant client driver that provides basic asynchronous communication support for Digi Edgeport USB-to-serial converters. Supported devices include Edgeport/1, Edgeport/2, Edgeport/21, Edgeport/4, Edgeport/421, Edgeport/8, and Edgeport/416. Serial device streams are built with appropriate modules that are pushed atop the `usbser_edge` driver by the `autopush(1M)` facility.

The `usbser_edge` module supports the `termio(7I)` device control functions specified by flags in the `c_cflag` word of the `termios` structure, and by the `IGNBRK`, `IGNPAR`, `PARMRK` and `INPCK` flags in the `c_iflag` word of the `termios` structure. All other `termio(7I)` functions must be performed by STREAMS modules pushed atop the driver. When a device is opened, the `ldterm(7M)` and `ttcompat(7M)` STREAMS modules are automatically pushed on top of the stream, providing the standard `termio(7I)` interface.

Use device logical names `/dev/term/[0-9]*` to access the serial ports. These names are typically used to provide a logical access point for a dial-in line that is used with a modem.

To allow a single tty line to be connected to a modem and used for incoming and outgoing calls, a special feature is available that is controlled by the minor device number. By accessing through device logical name`/dev/cua/[0-9]*`, you can open a port without the Carrier Detect signal being asserted, either through hardware or an equivalent software mechanism. These devices are commonly known as dial-out lines.

Unlike onboard serial ports, the `usbser_edge` ports cannot serve as a local serial console.

Application Programming Interface A dial-in line can be opened only if the corresponding dial-out line is closed. A blocking `/dev/term` open waits until the `/dev/cua` line is closed (which drops Data Terminal Ready, after which Carrier Detect usually drops as well) and carrier is detected again. A non-blocking`/dev/term` open returns an error if the `/dev/cua` is open.

If the `/dev/term` line is opened successfully (usually only when carrier is recognized on the modem), the corresponding `/dev/cua` line cannot be opened. This allows a modem and port to be used for dial-in (by enabling the line for login in `/etc/inittab`) or dial-out (by `tip(1)`, or `uucp(1C)`) when no one is logged in on the line.

Device hot-removal is functionally equivalent to modem disconnect event, as defined in `termio(7I)`.

ioctls The `usbser_edge` driver supports the standard set of `termio(7I)` ioctl calls.

Input and output line speeds can be set to the following baud rates: 0, 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200, or 230400. Input and output line speeds cannot be set independently; for example, when the output speed is set, the input speed is automatically set to the same speed.

Errors An `open()` fails under the following conditions:

`ENXIO` The unit being opened does not exist.

`EBUSY` The `/dev/cua` (dial-out) device is being opened while the/dev/term (dial-in device) is open, or the dial-in device is being opened with a no-delay open while the dial-out device is open.

`EBUSY` The unit has been marked as exclusive-use by another process with a `TIOCEXCL ioctl()` call.

`EIO` USB device I/O error.

Files

`/kernel/drv/usbser_edge`	32–bit x86 ELF kernel module
`/kernel/drv/amd64/usbser_edge`	64–bit x86 ELF kernel module
`/kernel/drv/sparcv9/usbser_edge`	64–bit SPARC ELF kernel module
`/kernel/drv/usbser_edge.conf`	configures communication mode
`/dev/cua/[0-9]*`	dial-out tty lines
`/dev/term/[0-9]*`	dial-in tty lines

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86, PCI-based systems
Availability	SUNWuedg

See Also `strconf(1)`, `tip(1)`, `uucp(1C)`, `autopush(1M)`, `ioctl(2)`, `open(2)`, `termios(3C)`, `attributes(5)`, `usba(7D)`, `termio(7I)`, `ldterm(7M)`, `ttcompat(7M)`

Diagnostics In addition to being logged, the following messages may appear on the system console. All messages are formatted in the following manner:

```
Warning: <device path> (usbser_edge<instance num>): Error Message...
```

Device was disconnected while open. Data may have been lost.
The device was hot-removed or powered off while it was open and a possible data transfer was in progress.

Device is not identical to the previous one on this port. Please disconnect and reconnect.
The device was hot-removed while open. A new device was hot-inserted which is not identical to the original device. Please disconnect the device and reconnect the original device to the same port.

Device has been reconnected, but data may have been lost.
The device that was hot-removed from its USB port has been re-inserted again to the same port. It is available for access but data from a previous transfer may be lost.

Cannot access <device>. Please reconnect.
This device has been disconnected because a device other than the original one has been inserted. The driver informs you of this fact by displaying the name of the original device.

The following messages may be logged into the system log. They are formatted in the following manner:

```
<device path><usbser_edge<instance number>): message...
```

Input overrun Data was lost.

Name usbsksp – Keyspan USB to serial converter driver

Synopsis

```
#include <fcntl.h>

#include <sys/termio.h>

usbskpsp@unit
```

Description The `usbsksp` driver is a loadable STREAMS and USBA (Solaris USB architecture) compliant client driver that provides basic asynchronous communication support for Keyspan USB-to-serial converters. The `usbsksp` driver supports the Keyspan USA19HS, USA49WG and USA49WLC models. By default, the USA19HS and USA49WG models are compatible with the `usbsksp` driver and no configuration or installation steps are required. (The USA49WG model is a USB 2.0 device comforming to *Universal Serial Bus Specification 2.0* and the USB 2.0 host controller is required to support the USA49WG model. Note that the USA49WG is not compatible with USB 1.1 host controllers). If you use the Keyspan USA49WLC model, you must download and install a firmware package to enable the device to work with the `usbsksp` driver. See the Keyspan website (*http://www.keyspan.com/downloads/sun/*) for more information. Serial device streams are built with appropriate modules that are pushed atop the `usbsksp` driver by the `autopush(1M)` facility.

The `usbsksp` module supports the `termio(7I)` device control functions specified by flags in the `c_cflag` word of the termios structure, and by the IGNBRK, IGNPAR, PARMRK and INPCK flags in the `c_iflag` word of the termios structure. All other `termio(7I)` functions must be performed by STREAMS modules pushed atop the driver. When a device is opened, the `ldterm(7M)` and `ttcompat(7M)` STREAMS modules are automatically pushed on top of the stream, providing the standard `termio(7I)` interface.

Use device logical names `/dev/term/[0-9]*` to access the serial ports. These names are typically used to provide a logical access point for a dial-in line that is used with a modem.

A special feature (controlled by the minor device number) is available that enables a single tty line to be connected to a modem and used for incoming and outgoing calls. By accessing through device logical name `/dev/cua/[0-9]*`, you can open a port without the carrier detect signal being asserted, either through hardware or an equivalent software mechanism. These devices are commonly known as 'dial-out' lines.

Unlike onboard serial ports, the `usbsksp` ports cannot serve as a local serial console.

Application Programming Interface A dial-in line can be opened only if the corresponding dial-out line is closed. A blocking `/dev/term` open waits until the `/dev/cua` line is closed (which drops Data Terminal Ready, after which Carrier Detect usually drops as well) and carrier is detected again. A non-blocking `/dev/term` open returns an error if the `/dev/cua` is open.

If the `/dev/term` line is opened successfully (usually only when carrier is recognized on the modem), the corresponding `/dev/cua` line cannot be opened. This allows a modem and port

to be used for dial-in (by enabling the line for login in `/etc/inittab`) or dial-out (by `tip(1)`, or `uucp(1C)`) when no one is logged in on the line.

Device hot-removal is functionally equivalent to a modem disconnect event, as defined in `termio(7I)`.

ioctls The `usbsksp` driver supports the standard set of `termio(7I)` ioctl calls.

Input and output line speeds can be set to the following baud rates: 0, 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200, or 230400. Input and output line speeds cannot be set independently. For example, when the output speed is set, the input speed is automatically set to the same speed.

Errors An `open()` fails under the following conditions:

`ENXIO` The unit being opened does not exist.

`EBUSY` The `/dev/cua` (dial-out) device is being opened while the `/dev/term` (dial-in device) is open, or the dial-in device is being opened with a no-delay open while the dial-out device is open.

`EBUSY` The unit has been marked as exclusive-use by another process with a TIOCEXCL ioctl() call.

`EIO` USB device I/O error.

Files

`/kernel/drv/usbsksp`	32-bit x86 ELF kernel module.
`/kernel/drv/amd64/usbsksp`	64-bit x86 ELF kernel module.
`/kernel/drv/sparcv9/usbsksp`	64-bit SPARC ELF kernel module.
`/dev/cua/[0-9]*`	dial-out tty lines.
`/dev/term/[0-9]*`	dial-in tty lines.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86, PCI-based systems
Availability	SUNWuksp

See Also `strconf(1)`, `tip(1)`, `uucp(1C)`, `autopush(1M)`, `ioctl(2)`, `open(2)`, `termios(3C)`, `attributes(5)`, `usba(7D)`, `termio(7I)`, `ldterm(7M)`, `ttcompat(7M)`

Diagnostics In addition to being logged, the following messages may appear on the system console. All messages are formatted in the following manner:

```
Warning: <device path> (usbsksp<instance num>): Error Message...
```

Device was disconnected while open. Data may have been lost.
The device has been hot-removed or powered off while it was open and a possible data transfer was in progress. The job may be aborted.

Device is not identical to the previous one on this port. Please disconnect and reconnect.
The device was hot-removed while open. A new device was hot-inserted which is not identical to the original device. Please disconnect the device and reconnect the original device to the same port.

Device has been reconnected, but data may have been lost.
The device that was hot-removed from its USB port has been re-inserted again to the same port. It is available for access but data from a previous transfer may be lost.

Cannot access <*device*>. Please reconnect.
This device has been disconnected because a device other than the original one has been inserted. The driver informs you of this fact by displaying the name of the original device.

No valid firmware available for Keyspan usa49wlc usb-to-serial adapter. Please download it from Keyspan website and install it.
By default, only an empty firmware package is installed for the usa49wlc model. Please download the SUNWukspfw package from Keyspan's web site and install it.

The following messages may be logged into the system log. They are formatted in the following manner:

```
<device path><usbsksp<instance number>): message...
```

Input overrun Data was lost.

Name usbsprl – Prolific PL2303 USB to serial converter driver

Synopsis

```
#include <fcntl.h>

#include <sys/termio.h>

usbsprl@unit
```

Description The `usbsprl` driver is a loadable STREAMS and USBA (Solaris USB architecture) compliant client driver that provides basic asynchronous communication support for Prolific PL2303 USB-to-serial converters. Supported devices include PL2303H, PL2303HX and PL2303X. Serial device streams are built with appropriate modules that are pushed atop the `usbsprl` driver by the `autopush`(1M) facility.

The `usbsprl` module supports the `termio`(7I) device control functions specified by flags in the `c_cflag` word of the termios structure, and by the IGNBRK, IGNPAR, PARMRK and INPCK flags in the `c_iflag` word of the termios structure. All other `termio`(7I) functions must be performed by STREAMS modules pushed atop the driver. When a device is opened, the `ldterm`(7M) and `ttcompat`(7M) STREAMS modules are automatically pushed on top of the stream, providing the standard `termio`(7I) interface.

Use device logical names `/dev/term/[0-9]*` to access the serial ports. These names are typically used to provide a logical access point for a dial-in line that is used with a modem.

A special feature (controlled by the minor device number) is available that enables a single tty line to be connected to a modem and used for incoming and outgoing calls. By accessing through device logical name `/dev/cua/[0-9]*`, you can open a port without the carrier detect signal being asserted, either through hardware or an equivalent software mechanism. These devices are commonly known as 'dial-out' lines.

Application Programming Interface A dial-in line can be opened only if the corresponding dial-out line is closed. A blocking `/dev/term` open waits until the `/dev/cua` line is closed (which drops Data Terminal Ready, after which Carrier Detect usually drops as well) and carrier is detected again. A non-blocking `/dev/term` open returns an error if the `/dev/cua` is open.

If the `/dev/term` line is opened successfully (usually only when carrier is recognized on the modem), the corresponding `/dev/cua` line cannot be opened. This allows a modem and port to be used for dial-in (by enabling the line for login in `/etc/inittab`) or dial-out (by `tip`(1), or `uucp`(1C)) when no one is logged in on the line.

Device hot-removal is functionally equivalent to a modem disconnect event, as defined in `termio`(7I).

ioctls The `usbsprl` driver supports the standard set of `termio`(7I) ioctl calls.

Input and output line speeds can be set to the following baud rates: 75, 150, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400 or 460800. Input and output line speeds cannot be set independently. For example, when the output speed is set, the input speed is automatically set to the same speed.

Errors An `open()` fails under the following conditions:

`ENXIO` The unit being opened does not exist.

`EBUSY` The `/dev/cua` (dial-out) device is being opened while the `/dev/term` (dial-in device) is open, or the dial-in device is being opened with a no-delay open while the dial-out device is open.

`EBUSY` The unit has been marked as exclusive-use by another process with a TIOCEXCL ioctl() call.

`EIO` USB device I/O error.

Files

`/kernel/drv/usbsprl`	32-bit x86 ELF kernel module.
`/kernel/drv/amd64/usbsprl`	64-bit x86 ELF kernel module.
`/kernel/drv/sparcv9/usbsprl`	64-bit SPARC ELF kernel module.
`/dev/cua/[0-9]*`	dial-out tty lines.
`/dev/term/[0-9]*`	dial-in tty lines.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86, PCI-based systems
Availability	SUNWuprl

See Also `strconf(1)`, `tip(1)`, `uucp(1C)`, `autopush(1M)`, `ioctl(2)`, `open(2)`, `termios(3C)`, `attributes(5)`, `usba(7D)`, `termio(7I)`, `ldterm(7M)`, `ttcompat(7M)`

Diagnostics In addition to being logged, the following messages may appear on the system console. All messages are formatted in the following manner:

```
Warning: <device path> (usbsprl<instance num>): Error Message...
```

Device was disconnected while open. Data may have been lost.
The device has been hot-removed or powered off while it was open and a possible data transfer was in progress. The job may be aborted.

Device is not identical to the previous one on this port. Please disconnect and reconnect.
The device was hot-removed while open. A new device was hot-inserted which is not identical to the original device. Please disconnect the device and reconnect the original device to the same port.

Device has been reconnected, but data may have been lost.
The device that was hot-removed from its USB port has been re-inserted again to the same port. It is available for access but data from a previou transfer may be lost.

Cannot access <*device*>. Please reconnect.
This device has been disconnected because a device other than the original one has been inserted. The driver informs you of this fact by displaying the name of the original device.

The following messages may be logged into the system log. They are formatted in the following manner:

```
<device path><usbsprl<instance number>): message...
```

Input overrun.
Data was lost.

Name usbvc – USB video class driver

Synopsis

```
#include <sys/usb/clients/video/usbvc/usbvc.h>

#include <sys/videodev2.h>

usbvc@unit-address
```

Description The `usbvc` driver is a USBA (Solaris USB Architecture)-compliant client driver that supports the *USB Device Class Definition for Video Devices* specification, Versions 1.0 and 1.1. The `usbvc` driver supports a subset of the video controls and formats described in the USB specification.

The `usbvc` driver also implements the Video4Linux2 API (*V4L2*), Version 0.20 for applications. For more information on the *V4L2 API*, visit *http://www.thedirks.org/v4l2*.

Note that the `usbvc` driver supports the video capture function only and that video output is not supported. For more information on supported USB video-class devices and functions, visit *http://www.sun.com/io*.

Reading Data The `usbvc` driver reads video data from the isochronous endpoint of the device. Bulk data endpoints are not supported.

MJPEG and UNCOMPRESSED video formats are supported. Isochronous data are read from the isochronous input device frame-by-frame and are maintained in a buffer array within the driver. Video frames are read from the driver using the `read(2)` or `mmap(2)` I/O method. For `read(2)`, each read returns a buffer of a video frame. For `mmap(2)`, each `VIDIOC_DQBUF` ioctl returns the buffer structure v4l2_buffer. (A video frame buffer pointer is included in the structure). See the *V4L2 API* for buffer structure and other related data structure information.

ioctls A brief overview of supported ioctl requests appears below. For more detailed information, refer to the *V4L2 API* document. Note: ioctl information presented in the *V4L2 API* document may differ slightly from the content of this manpage. In such cases, you should rely on the information in this manpage.

VIDIOC_QUERYCAP
: Query the device capabilities. Besides device capabilities, the `usbvc` driver returns structure v4l2_capability which includes information on the driver, data bus and OS kernel. Please note that the "Version" structure member has no meaning in Solaris and is always set to 1.

VIDIOC_ENUM_FMT
: Enumerate the video formats supported by the device.

VIDIOC_S_FMT
: Set a video format.

VIDIOC_G_FMT
: Get a video format.

VIDIOC_REQBUFS
Request the usbvc driver to allocate video data buffers. If a buffer is set to zero, the driver stops reading video data from the device and releases all allocated buffers. (For mmap(2) only).

VIDIOC_QUERYBUF
Query a given buffer's status. (For mmap(2) only).

VIDIOC_QBUF
Enqueue an empty buffer to the video data buffer array. (For mmap(2) only).

VIDIOC_DQBUF
Dequeue a done buffer from the video data buffer array. (For mmap(2) only).

VIDIOC_STREAMON
Start reading video data.

VIDIOC_STREAMOFF
Stop reading video data.

VIDIOC_ENUMINPUT
Enumerate all device inputs. Currently, the usbvc driver supports one input only.

VIDIOC_G_INPUT
Get the device's current input. At this time, the usbvc driver supports one input only.

VIDIOC_S_INPUT
Set the device's current input. At this time, the usbvc driver supports one input only.

VIDIOC_QUERYCTRL
Query the device and driver for supported video controls. Currently, the usbvc driver supports the brightness, contrast, saturation, hue, and gamma video controls.

VIDIOC_G_CTRL
Get the device's current video control.

VIDIOC_S_CTRL
Set the device's current video control.

VIDIOC_G_PARM
Get streaming parameters, the number of frames per second and number of buffers used internally by driver in read/write mode.

VIDIOC_S_PARM
Set streaming parameters, the number of frames per second and number of buffers used internally by driver in read/write mode.

Errors

EBUSY	An open was attempted after the device has already been opened.
EINVAL	An unsupported ioctl is received or an ioctl is attempted with an out-of-range value.

EIO The driver received an unrecoverable device error or the device did not respond or the device stalled when attempting an access. A read(2) or ioctl(2) did not complete due to a peripheral access.

ENXIO The driver received an open(2) request for a device for which the attach failed.

ENODEV The driver received an open(2) request for a disconnected device.

Files

`/kernel/drv/usbvc`
32-bit ELF kernel module. (x86)

`/kernel/drv/amd64/usbvc`
64-bit ELF kernel module. (x86)

`/kernel/drv/sparcv9/usbvc`
64-bit ELF kernel module. (SPARC)

`/dev/usb/*/*/*`
ugen(7D) nodes.

`/dev/videoN`
Device node for isochronous input from USB video device and device control.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86, PCI-based systems
Availability	SUNWusbvc

See Also cfgadm_usb(1M), ioctl(2), open(2), mmap(2), read(2), libusb(3LIB), attributes(5),ugen(7D), usba(7D), attach(9E)

Writing Device Drivers

System Administration Guide: Basic Administration

Universal Serial Bus Specification 1.0, 1.1 and 2.0— 1996, 1998, 2000

USB Device Class Definition for Video Devices 1.0 and 1.1— 2003, 2005

Video4Linux2 API (V4L2), Version 0.20

I/O Technologies and Solutions —http://www.sun.com/io

http://docs.sun.com

http://www.usb.org

http://www.thedirks.org/v4l2

Diagnostics In addition to being logged, the following messages may appear on the system console. All messages are formatted in the following manner:

```
Warning: <device path> (usbvc<instance num>):Error Message...
```

Device was disconnected while open. Data may have been lost.
The device has been hot-removed or powered off while it was open and a possible data transfer was in progress. The job may be aborted.

Cannot access <device>. Please reconnect.
This device has been disconnected because a device other than the original one has been inserted. The driver informs you of this fact by displaying the name of the original device.

Device is not identical to the previous one on this port. Please disconnect and reconnect.
The device was hot-removed while open. A new device was hot-inserted which is not identical to the original device. Please disconnect the device and reconnect the original device to the same port.

Notes The USB video device will be power-managed when the device is idle.

If a USB video device is hot-removed while active, a console warning is displayed requesting you to put the device back in the same port and telling you of potential data loss. Hot-removal of an active video device is strongly discouraged.

Always close all applications before hot-removing or hot-inserting a device. If an application is open when a device is hot-removed, inserting the device in a different port will create new `/dev/video`*N* links. Moving an active device to another port is not recommended.

Name usbwcm – STREAMS module for Wacom USB Tablets

Synopsis

```
# include <sys/usb/clients/usbinput/usbwcm/usbwcm.h>
```

Description The `usbwcm` STREAMS module processes byte streams generated by a Wacom USB tablet.

The `usbwcm` module must be pushed on top of the HID class driver. See `hid(7D)`. The `usbwcm` module translates data from Wacom USB tablet into formatted events expected by `Wacom X.org XInput`driver. The event structure is the same as that of FreeBSD `uwacom` driver.

Event Structure The event_input structure is defined in:
`usr/include/sys/usb/clients/usbinput/usbwcm/usbwcm.h`

```
   struct event_input {
#if defined(_LP64) || defined(_I32LPx)
      struct timeval32 time;
#else
      struct timeval time;
#endif
      uint16_t type;
      uint16_t index;
      int32_t value;
   };
```

time The event's timestamp. When the event occurred. The timestamp is not defined to be meaningful except by being compared with other input event timestamps.

type The event's unique *type*: button, relative/absolute valuator, sync, and so forth. *type* is among the following: `EVT_SYN`, `EVT_BTN`, `EVT_REL`, `EVT_ABS` and `EVT_MSC`.

index The event's *sub-type*. The index in a button event identifies which button status was changed. Typical button index includes: `BTN_LEFT`, `BTN_RIGHT`, `BTN_MIDDLE`, `BTN_SIDE`, `BTN_EXTRA`, `BTN_TOOL_PEN`, `BTN_TOOL_ERASER`, `BTN_TOOL_PAD`, `BTN_TOOL_MOUSE`, `BTN_TIP`, `BTN_STYLUS_1`, `BTN_STYLUS_2`

For absolute valuators, *index* is among the following: `ABS_X`, `ABS_Y`, `ABS_Z`, `ABS_RX`, `ABS_RY`, `ABS_WHEEL`, `ABS_PRESSURE`, `ABS_DISTANCE`, `ABS_TILT_X`, `ABS_TILT_Y`, and `ABS_MISC`.

Ioctls `EVTIOCGVERSION` The argument is a pointer to an int. This option returns the current version of the event interface implemented by the STREAMS module.

`EVTIOCGDEVID` The argument is a pointer to `event_dev_id` structure. This ioctl returns the identifiers of the device.

```
struct event_dev_id {
uint16_t bus;
#defineID_BUS_USB 3
uint16_t vendor;
uint16_t product;
```

```
uint16_t version;
};
```

EVTIOCGBM — The argument is a pointer to an variable-length char array. This ioctl returns the event types reported by device:

```
EVT_SYN, EVT_BTN, EVT_REL, EVT_ABS, EVT_MSC
```

EVTIOCGABS — The argument is a pointer to an `event_abs_axis` structure. This ioctl returns the ranges, and other parameters for the specified axis.

```
struct event_abs_axis {
int32_t value;
int32_t min;
int32_t max;
int32_t fuzz;
int32_t flat;
};
```

Files

`/kernel/strmod/usbwcm`	32-bit ELF kernel STREAMS module
`/kernel/strmod/amd64/usbwcm`	64-bit ELF kernel STREAMS module
`/kernel/strmod/sparcv9/usbwcm`	SPARC 64-bit ELF kernel STREAMS module

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86-based systems
Availability	SUNWusb

See Also `ioctl(2)`, `attributes(5)`, `hid(7D)`

System Administration Guide: Basic Administration

http://linuxwacom.sourceforge.net

http://www.sun.com

Name uscsi – user SCSI command interface

Synopsis

```
#include <sys/scsi/impl/uscsi.h>

ioctl(int fildes, int request, struct uscsi_cmd *cmd);
```

Description The `uscsi` command is very powerful and somewhat dangerous; therefore it has some permission restrictions. See `WARNINGS` for more details.

Drivers supporting this `ioctl(2)` provide a general interface allowing user-level applications to cause individual SCSI commands to be directed to a particular SCSI or ATAPI device under control of that driver. The `uscsi` command is supported by the `sd` driver for SCSI disks and ATAPI CD-ROM drives, and by the `st` driver for SCSI tape drives. `uscsi` may also be supported by other device drivers; see the specific device driver manual page for complete information.

Applications must not assume that all Solaris disk device drivers support the `uscsi` ioctl command. The SCSI command may include a data transfer to or from that device, if appropriate for that command. Upon completion of the command, the user application can determine how many bytes were transferred and the status returned by the device. Also, optionally, if the command returns a Check Condition status, the driver will automatically issue a Request Sense command and return the sense data along with the original status. See the `USCSI_RQENABLE` flag below for this Request Sense processing. The `uscsi_cmd` structure is defined in `<sys/scsi/impl/uscsi.h>` and includes the following members:

```
int uscsi_flags;              /* read, write, etc. see below */
short uscsi_status;          /* resulting status */
short uscsi_timeout;         /* Command Timeout */
caddr_t uscsi_cdb            /* CDB to send to target */
caddr_t uscsi_bufaddr;       /* i/o source/destination */
size_t uscsi_buflen;         /* size of i/o to take place*/
size_t uscsi_resid;          /* resid from i/o operation */
uchar_t uscsi_cdblen;        /* # of valid CDB bytes */
uchar_t uscsi_rqlen;         /* size of uscsi_rqbuf */
uchar_t uscsi_rqstatus;      /* status of request sense cmd */
uchar_t uscsi_rqresid;       /* resid of request sense cmd */
caddr_t uscsi_rqbuf;         /* request sense buffer */
void *uscsi_reserved_5;      /* Reserved for future use */
```

The fields of the `uscsi_cmd` structure have the following meanings:

`uscsi_flags`	The I/O direction and other details of how to carry out the SCSI command. Possible values are described below.
`uscsi_status`	The SCSI status byte returned by the device is returned in this field.
`uscsi_timeout`	Time in seconds to allow for completion of the command.
`uscsi_cdb`	A pointer to the SCSI CDB (command descriptor block) to be transferred to the device in command phase.

`uscsi_bufaddr`	The user buffer containing the data to be read from or written to the device.
`uscsi_buflen`	The length of `uscsi_bufaddr`.
`uscsi_resid`	If a data transfer terminates without transferring the entire requested amount, the remainder, or residue, is returned in this field.
`uscsi_cdblen`	The length of the SCSI CDB to be transferred to the device in command phase.
`uscsi_rqlen`	The length of `uscsi_rqbuf`, the application's Request Sense buffer.
`uscsi_rqstatus`	The SCSI status byte returned for the Request Sense command executed automatically by the driver in response to a Check Condition status return.
`uscsi_rqresid`	The residue, or untransferred data length, of the Request Sense data transfer (the number of bytes, less than or equal to `uscsi_rqlen`, which were not filled with sense data).
`uscsi_rqbuf`	Points to a buffer in application address space to which the results of an automatic Request Sense command are written.
`uscsi_reserved_5`	Reserved for future use.

The `uscsi_flags` field defines the following:

```
USCSI_WRITE                 /* send data to device */
USCSI_SILENT                /* no error messages */
USCSI_DIAGNOSE              /* fail if any error occurs */
USCSI_ISOLATE               /* isolate from normal commands */
USCSI_READ                  /* get data from device */
USCSI_ASYNC                 /* set bus to asynchronous mode */
USCSI_SYNC                  /* return bus to sync mode if possible */
USCSI_RESET                 /* reset target */
USCSI_RESET_TARGET          /* reset target */
USCSI_RESET_LUN             /* reset logical unit */
USCSI_RESET_ALL             /* reset all targets */
USCSI_RQENABLE              /* enable request sense extensions */
USCSI_RENEGOT               /* renegotiate wide/sync on next I/O */
```

The `uscsi_flags` bits have the following interpretation:

`USCSI_WRITE`	Data will be written from the initiator to the target.
`USCSI_SILENT`	The driver should not print any console error messages or warnings regarding failures associated with this SCSI command.

USCSI_DIAGNOSE	The driver should not attempt any retries or other recovery mechanisms if this SCSI command terminates abnormally in any way.
USCSI_ISOLATE	This SCSI command should not be executed with other commands.
USCSI_READ	Data will be read from the target to the initiator.
USCSI_ASYNC	Set the SCSI bus to asynchronous mode before running this command.
USCSI_SYNC	Set the SCSI bus to synchronous mode before running this command.
USCSI_RESET	Send a SCSI bus device reset message to this target.
USCSI_RESET_TARGET	Same as USCSI_RESET. Use this flag to request TARGET RESET. (USCSI_RESET is maintained only for compatibility with old applications).
USCSI_RESET_LUN	Send a SCSI logical unit reset message to this target.
USCSI_RESET_ALL	USCSI_RESET_ALL, USCSI_RESET/USCSI_RESET_TARGET and USCSI_RESET_LUN are mutually exclusive options and issuing them in any simultaneous combination will result in implementation-dependent behavior When a USCSI reset request is combined with other SCSI commands, the following semantics take effect: If the USCSI RESET flag is specified, the other fields (other than uscsi_flags) in the uscsi_cmd are ignored. The uscsi_cdblen *must* be set to zero.
USCSI_RQENABLE	Enable Request Sense extensions. If the user application is prepared to receive sense data, this bit must be set, the fields uscsi_rqbuf and uscsi_rqbuflen must be non-zero, and the uscsi_rqbuf must point to memory writable by the application.
USCSI_RENEGOT	Tells USCSI to renegotiate wide mode and synchronous transfer speed before the transmitted SCSI command is executed. This flag in effects tells the target driver to pass the FLAG_RENEGOTIATE_WIDE_SYNC flag in the SCSI packet before passing the command to an adapter driver for transport. See the scsi_pkt(9S) flag FLAG_RENEGOTIATE_WIDE_SYNC for more information.

ioctls The `ioctl` supported by drivers providing the `uscsi` interface is:

`USCSICMD` The argument is a pointer to a `uscsi_cmd` structure. The SCSI device addressed by that driver is selected, and given the SCSI command addressed by `uscsi_cdb`. If this command requires a data phase, the `uscsi_buflen` and `uscsi_bufaddr` fields must be set appropriately; if data phase occurs, the `uscsi_resid` is returned as the number of bytes not transferred. The status of the command, as returned by the device, is returned in the `uscsi_status` field. If the command terminates with Check Condition status, and Request Sense is enabled, the sense data itself is returned in `uscsi_rqbuf`. The `uscsi_rqresid` provides the residue of the Request Sense data transfer.

Errors

`EINVAL` A parameter has an incorrect, or unsupported, value.

`EIO` An error occurred during the execution of the command.

`EPERM` A process without root credentials tried to execute the `USCSICMD` ioctl.

`EFAULT` The `uscsi_cmd` itself, the `uscsi_cdb`, the `uscsi_buf`, or the `uscsi_rqbuf` point to an invalid address.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWhea
Interface Stability	Committed

See Also `ioctl(2)`, `attributes(5)`, `sd(7D)`, `st(7D)`

ANSI Small Computer System Interface-2 (SCSI-2)

Warnings The `uscsi` command is very powerful, but somewhat dangerous, and so its use is restricted to processes running as root, regardless of the file permissions on the device node. The device driver code expects to own the device state, and `uscsi` commands can change the state of the device and confuse the device driver. It is best to use `uscsi` commands only with no side effects, and avoid commands such as Mode Select, as they may cause damage to data stored on the drive or system panics. Also, as the commands are not checked in any way by the device driver, any block may be overwritten, and the block numbers are absolute block numbers on the drive regardless of which slice number is used to send the command.

The `uscsi` interface is not recommended for very large data transfers (typically more than 16MB). If the requested transfer size exceeds the maximum transfer size of the DMA engine, it will not be broken up into multiple transfers and DMA errors may result.

The `USCSICMD` ioctl associates a `struct uscsi_cmd` with a device by using an open file descriptor to the device. Other APIs might provide the same `struct uscsi_cmd` programming interface, but perform device association in some other manner.

Name usoc – universal serial optical controller for Fibre Channel arbitrated loop (SOC+) device driver

Description The Fibre Channel adapter is an SBus card that implements two full duplex Fibre Channel interfaces. Each interface can connect to a Fibre Channel arbitrated loop (FC-AL). The usoc device driver is a nexus driver and implements portions of the FC-2 and FC-4 layers of FC-AL.

Files `/kernel/drv/usoc` 32–bit ELF kernel module

`/kernel/drv/sparcv9/usoc` 64–bit ELF kernel module

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC
Interface stability	Unknown
Availability	SUNWusoc

See Also `fctl`(7D), `sbus`(4), `fcp`(7D), `fp`(7d), `ssd`(7D)

Writing Device Drivers

Fibre Channel Physical and Signaling Interface (FC-PH) ANSI X3.230: 1994

Fibre Channel Arbitrated Loop (FC-AL) ANSI X3.272-1996

Fibre Channel Private Loop SCSI Direct Attach (FC-PLDA) NCITS TR-19:1998

Fabric Channel Loop Attachment (FC-FLA), NCITS TR-20:1998

Diagnostics The following messages are logged and may also appear on the system console. On the console these messages are preceded by:

`usoc%d:`

where

`usoc%d:`

is the per-port instance number of the `usoc` controller.

`Fibre Channel is ONLINE`

The Fibre Channel loop is now online.

`Fibre Channel Loop is ONLINE`

The Fibre Channel loop is now online.

`Fibre Channel Loop is OFFLINE`

The Fibre Channel loop is now offline.

```
attach failed: device in slave-only slot.
```

Move soc+ card to another slot.

```
attach failed: alloc soft state.
```

Driver did not attach, devices will be inaccessible.

```
attach failed: bad soft state.
```

Driver did not attach, devices will be inaccessible.

```
attach failed: unable to map eeprom
```

Driver was unable to map device memory; check for bad hardware. Driver did not attach to device, devices will be inaccessible.

```
attach failed: unable to map XRAM
```

Driver was unable to map device memory; check for bad hardware. Driver did not attach to device, devices will be inaccessible.

```
attach failed: unable to map registers
```

Driver was unable to map device registers; check for bad hardware. Driver did not attach to device, devices will be inaccessible.

```
attach failed: unable to access status register
```

Driver was unable to map device registers; check for bad hardware. Driver did not attach to device, devices will be inaccessible.

```
attach failed: unable to install interrupt handler
```

Driver was not able to add the interrupt routine to the kernel. Driver did not attach to device, devices will be inaccessible.

```
attach failed: unable to access host adapter XRAM
```

Driver was unable to access device RAM; check for bad hardware. Driver did not attach to device, devices will be inaccessible.

```
attach failed: unable to write host adapter XRAM
```

Driver was unable to write device RAM; check for bad hardware. Driver did not attach to device, devices will be inaccessible.

```
attach failed: read/write mismatch in XRAM
```

Driver was unable to verify device RAM; check for bad hardware. Driver did not attach to device, devices will be inaccessible.

Name uwba, uwb – Solaris UWB Architecture (UWBA)

Description Ultra-WideBand (UWB) radio technology supports high bandwidth for wireless devices. UWBA is a miscellaneous module and it supports radio controller drivers for UWB based devices like HWA (Host Wire Adapter), WHCI (Wireless Host Controller Interface) and so forth. For example, both HWA radio controller driver (`hwarc`) and `whci` driver register to `uwba` during attach.

UWBA provides a series of common interfaces for drivers that support UWB radio technology. Each radio controller driver register itself as a UWB dev to the uwba model in the attach entry, then other driver or module can control this device to perform the UWB functions through a list of common interface. For example, a `hwahc` driver can control the `hwarc` driver to scan in a specific channel, start/stop beacon, manage device/MAC address, and so forth.

Files

`/kernel/misc/uwba`	32-bit ELF 86 kernel module
`/kernel/misc/amd64/uwba`	64-bit x86 ELF kernel module
`/kernel/misc/sparcv9/uwba`	64-bit SPARC ELF kernel module

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	PCI-based systems
Availability	SUNWuwb

See Also `attributes(5)`, `hwahc(7D)`, `hwarc(7D)`, `usba(7D)`

Writing Device Drivers

ECMA-368 High Rate Ultra Wideband PHY and MAC Standard, 1st Edition

Wireless Host Controller Interface Specification for Certified Wireless Universal Serial Bus, Version 0.95

Wireless Universal Serial Bus Specification 1.0

Name virtualkm – Virtual keyboard and mouse

Synopsis

```
/dev/kbd

/dev/mouse

#include <sys/types.h>

#include <sys/kbio.h>

int ioctl(int fildes, int command, ... /*arg*/);
```

Description A virtual keyboard or mouse is an abstraction of one or more physical keyboards or mice (USB or PS2) connected to a system. Input streams for these physical devices are coalesced into a single input stream and appear as a single device to the upper layers.

`/dev/kbd` is the virtual keyboard device file. Inputs from multiple keyboards are coalesced into a single input stream, meaning that all keyboards appear as a single keyboard to a console or window system and accordingly, are treated as a single device. The virtual keyboard layout is consistent with the layout of the first keyboard plugged into the system. Note that on x86 platforms, the virtual keyboard layout can be overloaded by `eeprom(1M)`.

`/dev/mouse` is the virtual mouse device file. Inputs from multiple mice are coalesced into a single input stream, meaning that all mice appear as single mouse to the window system.

Commands from applications are dispatched by the virtual keyboard/mouse facility to the underlying physical devices and will succeed provided that one of the underlying devices responds with success. For example, a single command issued to turn on LED's will turn on corresponding LED's for all underlying physical keyboards.

Although physical keyboards/mice are linked to the virtual keyboard/mouse facility, each may be opened separately by accessing its associated device file. (For example, `/dev/usb/hid0` for a usb mouse). Directly accessing a device file can be useful for multi-seat, `gok(1)` or similar purposes.

When a single physical device is opened via its associated device file, it is automatically removed from the single virtual input stream. When closed, it is automatically re– coalesced into the single virtual input stream.

Under the `virtualkm` facility, the PS/2 mouse is coalesced into a virtual mouse single input stream and can be accessed using the `/dev/mouse` file. (Note that in previous releases, the PS/2 mouse was accessed via the `/dev/kdmouse` physical device file). In the current release, you use the `/dev/kdmouse` file to directly access the physical PS/2 mouse.

INTERFACES The virtual mouse provides the following event ID's for mouse capability changes:

MOUSE_CAP_CHANGE_NUM_BUT	This event is reported when the total number of mouse buttons changes. The `Firm_event.value` is set to the new button total, which is the maximum number of all mice buttons. Other fields are ignored.

	MOUSE_CAP_CHANGE_NUM_WHEEL	This event is reported when the total number of mouse wheels changes. The `Firm_event.value` is set to the new wheel total. Other fields are ignored. The event value (`Firm_event.value`) can be 0 (no wheel), 1 (vertical wheel), or 2 (vertical and horizontal wheel).

The `Firm_event` structure is described in <`sys/vuid_event.h`>. As with other events, firm events are received using `read(2)`.

Event ID's are used by applications (including certain mouse demo applications) that are programmed to graphically represent the actual number of buttons and wheels on a mouse. When an application of this type receives a `Firm_event` with a ID MOUSE_CAP_CHANGE_NUM_BUT or MOUSE_CAP_CHANGE_NUM_WHEEL event, it is instructed to update its state information using the new value. Consider, for example, a mouse demo application whose sole function is to display a mouse with buttons that graphically correspond to the actual number of buttons on the mouse. If, for example, the system has a single two-button USB mouse attached, the application, by default, will graphically display the mouse with a left and a right button. However, if a another three-button USB mouse is hot-plugged into the system, a MOUSE_CAP_CHANGE_NUM_BUT Firm event with `Firm_event.value` of three instructs the demo application to update the mouse display to indicate three buttons.

ioctls

KIOCSETFREQ — Sets the frequency for either keyboard beeper or console beeper. To set the corresponding beeper frequency, *arg* must point to a `freq_request` structure:

```
struct freq_request {
        enum fr_beep_type type; /* beep type */
        int16_t freq;           /* frequency */
};
```

Where *type* is the corresponding beeper type defined as:

```
enum fr_beep_type { CONSOLE_BEEP =1, KBD_BEEP =2 };
```

and *freq* is the frequency value to be set as the beeper frequency indicated by *type*. This value should be between 0 and 32767 with border inclusive.

Files

`/dev/kbd`	Virtual Keyboard device file.
`/dev/mouse`	Virtual Mouse device file.
`/dev/kdmouse`	Physical PS/2 mouse device file.
`/dev/usb/hid*`	Physical USB keyboard/mouse device file.
`/etc/dacf.conf`	Device auto-configuration file.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86
Availability	SUNWckr, SUNWcsd, SUNWusb, SUNWpsdcr, SUNWcakr.i
Interface Stability	Evolving

See Also kbd(1), eeprom(1M), read(2), attributes(5), hid(7D), usba(7D), kb(7M), usbkbm(7M), usbms(7M), vuidmice(7M)

See gok(1) in the GNOME man pages, available in the SUNWgnome package.

Diagnostics The messages described below may appear on the system console as well as being logged. All messages are formatted in the following manner:

```
WARNING: Error message...
```

conskbd: keyboard is not available for system debugging: *device_path*.
Errors were encountered while entering kmdb during initialization for debugger mode. As a result, the keyboard is not available.

conskbd: keyboard is not available: <*device_path*>
Errors were encountered while exiting kmdb during un-initialization for debugger mode. As a result, the keyboard is not available.

Failed to relink the mouse <*device_path*> underneath virtual mouse
An error was encountered and the mouse is unavailable. (When a mouse is physically opened via a physical device file such as `/dev/usb/hid0`, it is removed from the single virtual input stream (`/dev/mouse`). When closed, it is re-coalesced into a single virtual input stream beneath `/dev/mouse`. If an error is encountered, (for example, the mouse has been physically removed), it is unavailable beneath `/dev/mouse`.

Notes Currently, the `virtualkm` device supports only USB and PS2 keyboards and mice.

The `virtualkm` device maintains complete compatibility on select legacy systems, (including Ultra 10's), that are equipped with serial keyboard/mouse.

Name visual_io – Solaris VISUAL I/O control operations

Synopsis

```
#include <sys/visual_io.h>
```

Description The Solaris VISUAL environment defines a small set of ioctls for controlling graphics and imaging devices.

The `VIS_GETIDENTIFIER` ioctl is mandatory and must be implemented in device drivers for graphics devices using the Solaris VISUAL environment. The `VIS_GETIDENTIFIER` ioctl is defined to return a device identifier from the device driver. This identifier must be a uniquely-defined string.

There are two additional sets of ioctls. One supports mouse tracking via hardware cursor operations. Use of this set is optional, however, if a graphics device has hardware cursor support and implements these ioctls, the mouse tracking performance is improved. The remaining set supports the device acting as the system console device. Use of this set is optional, but if a graphics device is to be used as the system console device, it must implement these ioctls.

The VISUAL environment also defines interfaces for non-ioctl entry points into the driver that the Solaris operating environment calls when it is running in standalone mode (for example, when using a stand-alone debugger, entering the PROM monitor, or when the system panicking). These are also known as `Polled I/O` entry points, which operate under an explicit set of restrictions, described below.

ioctls

`VIS_GETIDENTIFIER` This `ioctl()` returns an identifier string to uniquely identify a device used in the Solaris VISUAL environment. This is a mandatory ioctl and must return a unique string. We suggest that the name be formed as <*companysymbol*><*devicetype*>.

`VIS_GETIDENTIFIER` takes a `vis_identifier` structure as its parameter. This structure has the form:

```
#define VIS_MAXNAMELEN 128
struct vis_identifier {
       char name[VIS_MAXNAMELEN];
};
```

`VIS_GETCURSOR`
`VIS_SETCURSOR` These ioctls fetch and set various cursor attributes, using the `vis_cursor` structure.

```
struct vis_cursorpos {
       short    x;         /* cursor x coordinate */
       short    y;         /* cursor y coordinate */
};

struct vis_cursorcmap {
      int    version;            /* version */
```

```
        int    reserved;
        unsigned char *red;  /* red color map elements */
        unsigned char *green;/* green color map elements */
        unsigned char *blue; /* blue color map elements */
};

#define VIS_CURSOR_SETCURSOR   0x01  /* set cursor */
#define VIS_CURSOR_SETPOSITION 0x02  /* set cursor position */
#define VIS_CURSOR_SETHOTSPOT  0x04  /* set cursor hot spot */
#define VIS_CURSOR_SETCOLORMAP 0x08  /* set cursor colormap */
#define VIS_CURSOR_SETSHAPE 0x10     /* set cursor shape */
#define VIS_CURSOR_SETALL    \
    (VIS_CURSOR_SETCURSOR | VIS_CURSOR_SETPOSITION | \
    VIS_CURSOR_SETHOTSPOT | VIS_CURSOR_SETCOLORMAP | \
    VIS_CURSOR_SETSHAPE)

struct vis_cursor {
    short set;                    /* what to set */
    short enable;                 /* cursor on/off */
    struct vis_cursorpos pos;     /* cursor position */
    struct  vis_cursorpos hot;    /* cursor hot spot */
    struct vis_cursorcmap cmap;   /* color map info */
    struct vis_cursorpos size;    /* cursor bitmap size */
    char  *image;                 /* cursor image bits */
    char  *mask;                  /* cursor mask bits */
};
```

The `vis_cursorcmap` structure should contain pointers to two elements, specifying the red, green, and blue values for foreground and background.

`VIS_SETCURSORPOS`
`VIS_MOVECURSOR` These ioctls fetch and move the current cursor position, using the `vis_cursorpos` structure.

Console Optional Ioctls

The following ioctl sets are used by graphics drivers that are part of the system console device. All of the ioctls must be implemented to be a console device. In addition, if the system does not have a prom or the prom goes away during boot, the special standalone ioctls (listed below) must also be implemented.

The coordinate system for the console device places 0,0 at the upper left corner of the device, with rows increasing toward the bottom of the device and columns increasing from left to right.

VIS_PUTCMAP
`VIS_GETCMAP` Set or get color map entries.

The argument is a pointer to a `vis_cmap` structure, which contains the following fields:

```
struct vis_cmap {
    int    index;
    int    count;
    uchar_t    *red;
    uchar_t    *green;
    uchar_t    *blue;
}
```

`index` is the starting index in the color map where you want to start setting or getting color map entries.

`count` is the number of color map entries to set or get. It also is the size of the `red`, `green`, and `blue` color arrays.

`*red`, `*green`, and `*blue` are pointers to unsigned character arrays which contain the color map info to set or where the color map info is placed on a get.

`VIS_DEVINIT` Initializes the graphics driver as a console device.

The argument is a pointer to a `vis_devinit` structure. The graphics driver is expected to allocate any local state information needed to be a console device and fill in this structure.

```
struct vis_devinit {
    int  version;
    screen_size_t  width;
    screen_size_t  height;
    screen_size_t  linebytes;
    unit_t    size;
    int       depth;
    short  mode;
    struct vis_polledio    *polledio;
    vis_modechg_cb_t       modechg_cb;
    struct vis_modechg_arg *modechg_arg;
};
```

`version` is the version of this structure and should be set to `VIS_CONS_REV`.

`width` and `height` are the width and height of the device. If `mode` (see below) is `VIS_TEXT` then `width` and `height` are the number of characters wide and high of the device. If `mode` is `VIS_PIXEL` then `width` and `height` are the number of pixels wide and high of the device.

`linebytes` is the number of bytes per line of the device.

`size` is the total size of the device in pixels.

`depth` is the pixel depth in device bits. Currently supported depths are: `1`, `4`, `8` and `24`.

`mode` is the mode of the device. Either `VIS_PIXEL` (data to be displayed is in bitmap format) or `VIS_TEXT` (data to be displayed is in ascii format).

`polledio` is used to pass the address of the structure containing the standalone mode polled I/O entry points to the device driver back to the terminal emulator. The `vis_polledio` interfaces are described in the Console Standalone Entry Points section of this manpage. These entry points are where the operating system enters the driver when the system is running in standalone mode. These functions perform identically to the VIS_CONSDISPLAY, VIS_CONSCURSOR and VIS_CONSCOPY ioctls, but are called directly by the Solaris operating environment and must operate under a very strict set of assumptions.

`modechg_cb` is a callback function passed from the terminal emulator to the framebuffer driver which the frame-buffer driver must call whenever a video mode change event occurs that changes the screen height, width or depth. The callback takes two arguments, an opaque handle, `modechg_arg`, and the address of a vis_devinit struct containing the new video mode information.

`modechg_arg` is an opaque handle passed from the terminal emulator to the driver, which the driver must pass back to the terminal emulator as an argument to the `modechg_cb` function when the driver notifies the terminal emulator of a video mode change.

`VIS_DEVFINI` Tells the graphics driver that it is no longer the system console device. There is no argument to this ioctl. The driver is expected to free any locally kept state information related to the console.

`VIS_CONSCURSOR` Describes the size and placement of the cursor on the screen. The graphics driver is expected to display or hide the cursor at the indicated position.

The argument is a pointer to a `vis_conscursor` structure which contains the following fields:

```
struct vis_conscursor {
    screen_pos_t   row;
    screen_pos_t   col;
    screen_size_t  width;
    screen_size_t  height
    color_t        fg_color;
    color_t        bg_color;
    short          action;
};
```

`row` and `col` are the first row and column (upper left corner of the cursor).

`width` and `height` are the width and height of the cursor.

If `mode` in the `VIS_DEVINIT` ioctl is set to `VIS_PIXEL`, then `col`, `row`, `width` and `height` are in pixels. If `mode` in the `VIS_DEVINIT` ioctl was set to `VIS_TEXT`, then `col`, `row`, `width` and `height` are in characters.

fg_color and bg_color are the foreground and background color map indexes to use when the action (see below) is set to VIS_DISPLAY_CURSOR.

action indicates whether to display or hide the cursor. It is set to either VIS_HIDE_CURSOR or VIS_DISPLAY_CURSOR.

VIS_CONSDISPLAY — Display data on the graphics device. The graphics driver is expected to display the data contained in the vis_display structure at the specified position on the console.

The vis_display structure contains the following fields:

```
struct vis_display {
    screen_pos_t   row;
    screen_pos_t   col;
    screen_size_t  width;
    screen_size_t  height;
    uchar_t        *data;
    color_t        fg_color;
    color_t        bg_color;
};
```

row and col specify at which starting row and column the date is to be displayed. If mode in the VIS_DEVINIT ioctl was set to VIS_TEXT, row and col are defined to be a character offset from the starting position of the console device. If mode in the VIS_DEVINIT ioctl was set to VIS_PIXEL, row and col are defined to be a pixel offset from the starting position of the console device.

width and height specify the size of the data to be displayed. If mode in the VIS_DEVINIT ioctl was set to VIS_TEXT, width and height define the size of data as a rectangle that is width characters wide and height characters high. If mode in the VIS_DEVINIT ioctl was set to VIS_PIXEL, width and height define the size of data as a rectangle that is width pixels wide and height pixels high.

*data is a pointer to the data to be displayed on the console device. If mode in the VIS_DEVINIT ioctl was set to VIS_TEXT, data is an array of ASCII characters to be displayed on the console device. The driver must break these characters up appropriately and display it in the retangle defined by row, col, width, and height. If mode in the VIS_DEVINIT ioctl was set to VIS_PIXEL, data is an array of bitmap data to be displayed on the console device. The driver must break this data up appropriately and display it in the retangle defined by row, col, width, and height.

The fg_color and bg_color fields define the foreground and background color map indexes to use when displaying the data. fb_color is used for on pixels and bg_color is used for off pixels.

VIS_CONSCOPY — Copy data from one location on the device to another. The driver is expected to copy the specified data. The source data should not be modified. Any modifications to the source data should be as a side effect of the copy destination overlapping the copy source.

The argument is a pointer to a vis_copy structure which contains the following fields:

```
struct vis_copy {
    screen_pos_t  s_row;
    screen_pos_t  s_col;
    screen_pos_t  e_row;
    screen_pos_t  e_col;
    screen_pos_t  t_row;
    screen_pos_t  t_col;
    short         direction;
};
```

s_row, s_col, e_row, and e_col define the source rectangle of the copy. s_row and s_col are the upper left corner of the source rectangle. e_row and e_col are the lower right corner of the source rectangle. If mode in the VIS_DEVINIT ioctl() was set to VIS_TEXT, s_row, s_col, e_row, and e_col are defined to be character offsets from the starting position of the console device. If mode in the VIS_DEVINIT ioctl was set to VIS_PIXEL, s_row, s_col, e_row, and e_col are defined to be pixel offsets from the starting position of the console device.

t_row and t_col define the upper left corner of the destination rectangle of the copy. The entire rectangle is copied to this location. If mode in the VIS_DEVINIT ioctl was set to VIS_TEXT, t_row, and t_col are defined to be character offsets from the starting position of the console device. If mode in the VIS_DEVINIT ioctl was set to VIS_PIXEL, t_row, and t_col are defined to be pixel offsets from the starting position of the console device.

direction specifies which way to do the copy. If direction is VIS_COPY_FORWARD the graphics driver should copy data from position (s_row, s_col) in the source rectangle to position (t_row, t_col) in the destination rectangle. If direction is VIS_COPY_BACKWARDS the graphics driver should copy data from position (e_row, e_col) in the source rectangle to position (t_row+(e_row-s_row), t_col+(e_col-s_col)) in the destination rectangle.

Console Standalone Entry Points (Polled I/O Interfaces)

Console standalone entry points are necessary only if the driver is implementing console-compatible extensions. All console vectored standalone entry points must be implemented along with all console-related ioctls if the console extension is implemented.

```
struct vis_polledio {
    struct vis_polledio_arg *arg;
    void    (*display)(vis_polledio_arg *, struct vis_consdisplay *);
    void    (*copy)(vis_polledio_arg *, struct vis_conscopy *);
    void    (*cursor)(vis_polledio_arg *, struct vis_conscursor *);
};
```

The `vis_polledio` structure is passed from the driver to the Solaris operating environment, conveying the entry point addresses of three functions which perform the same operations of their similarly named ioctl counterparts. The rendering parameters for each entry point are derived from the same structure passed as the respective ioctl. See the Console Optional Ioctls section of this manpage for an explanation of the specific function each of the entry points, display(), copy() and cursor() are required to implement. In addition to performing the prescribed function of their ioctl counterparts, the standalone vectors operate in a special context and must adhere to a strict set of rules. The polled I/O vectors are called directly whenever the system is quisced (running in a limited context) and must send output to the display. Standalone mode describes the state in which the system is running in single-threaded mode and only one processor is active. Solaris operating environment services are stopped, along with all other threads on the system, prior to entering any of the polled I/O interfaces. The polled I/O vectors are called when the system is running in a standalone debugger, when executing the PROM monitor (OBP) or when panicking.

The following restrictions must be observed in the polled I/O functions:

1. The driver must not allocate memory.
2. The driver must not wait on mutexes.
3. The driver must not wait for interrupts.
4. The driver must not call any DDI or LDI services.
5. The driver must not call any system services.

The system is single-threaded when calling these functions, meaning that all other threads are effectively halted. Single-threading makes mutexes (which cannot be held) easier to deal with, so long as the driver does not disturb any shared state. See *Writing Device Drivers* for more information about implementing polled I/O entry points.

See Also `ioctl(2)`

Writing Device Drivers

Notes On SPARC systems, compatible drivers supporting the kernel terminal emulator should export the *tem-support* DDI property.*tem-support* indicates that the driver supports the kernel terminal emulator. By exporting `tem-support` it's possible to avoid premature handling of an incompatible driver.

tem-support — This DDI property, set to 1, means driver is compatible with the console kernel framebuffer interface.

Name vni – STREAMS virtual network interface driver

Description The `vni` pseudo device is a multi-threaded, loadable, clonable, STREAMS pseudo-device supporting the connectionless Data Link Provider Interface `dlpi(7P)` Style 2. Note that DLPI is intended to interact with IP, meaning that DLPI access to applications is not supported. (For example, snoop fails on the `vni` interface.)

The `vni` device is a software-only interface and does not send or receive data. The device provides a DLPI upper interface that identifies itself to IP with a private media type. It can be configured via `ifconfig(1M)` and can have IP addresses assigned to it, making aliases possible.

The `vni` pseudo device is particularly useful in hosting an IP address when used in conjunction with the 'usesrc' `ifconfig` option (see `ifconfig(1M)` for examples). The logical instances of the device can also be used to host addresses as an alternative to hosting them over the loopback interface.

Multicast is not supported on this device. More specifically, the following options return an error when used with an address specified on `vni`: IP_MULTICAST_IF, IP_ADD_MEMBERSHIP, IP_DROP_MEMBERSHIP, IPV6_MULTICAST_IF, IPV6_JOIN_GROUP, IPV6_LEAVE_GROUP. In addition, broadcast is not supported.

Because there is no physical hardware configured below it, no traffic can be received through nor transmitted on a virtual interface. All packet transmission and reception is accomplished with existing physical interfaces and tunnels. Because applications that deal with packet transmission and reception (such as packet filters) cannot filter traffic on virtual interfaces, you cannot set up a packet filter on a virtual interface. Instead, you should configure the policy rules to apply to the physical interfaces and tunnels, and if necessary, use the virtual IP addresses themselves as part of the rule configuration. Also, note that the virtual interface cannot be part of an IP multipathing (IPMP) group.

Files `/dev/vni` 64–bit ELF kernel driver

See Also `ifconfig(1M)`, `in.mpathd(1M)`, `ip(7P)`, `ip6(7P)`

Name vr – driver for VIA Rhine fast Ethernet controllers

Description The `vr` Fast Ethernet driver is GLD based and supporting the VIA Rhine family of Fast Ethernet adapters:

```
pci1106,3043 VIA VT86C100A Fast Ethernet
pci1106,3065 VT6102 VIA Rhine II
pci1106,3106 VT6105 VIA Rhine III
pci1106,3053 VT6105 VIA Rhine III Management Adapter
```

The `vr` driver supports IEEE 802.3 auto-negotiation, flow control and VLAN tagging.

Configuration The default configuration is `autonegotiation` with bidirectional flow control. The advertised capabilities for `autonegotiation` are based on the capabilities of the PHY.

You can set the capabilities advertised by the `vr` controlled device using `dladm(1M)`. The driver supports a number of parameters, the names of which begin with `en_` (enabled). Each of these boolean parameters determines if the device advertises that mode of operation when the hardware supports it.

The `adv_autoneg_cap` parameter controls whether auto-negotiation is performed. If `adv_autoneg_cap` is 0, the driver selects the speed/duplex combination from the first non-zero parameter from this list:

```
en_100fdx_cap     100Mpbs full duplex
en_100hdx_cap     100Mpbs half duplex
en_10fdx_cap      10Mpbs full duplex
en_10hdx_cap      10Mpbs half duplex
```

All capabilities default to enabled. Changing any capability parameter causes the link to go down while the link partners renegotiate the link using the newly changed capabilities.

Limitations The `vr` driver does not support asymmetric `flowcontrol`. VT86C100A and Rhine II adapters are not capable of transmitting `flowcontrol` messages

Files

`/dev/vr`	Special character device
`/kernel/drv/vr`	32-bit device driver (x86)
`/kernel/drv/sparcv9/vr`	64-bit device driver (SPARC)
`/kernel/drv/amd64/vr`	64-bit device driver (x86)

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86

See Also dladm(1M), netstat(1M), driver.conf(4), attributes(5), ieee802.3(5), dlpi(7P), streamio(7I)

Writing Device Drivers

STREAMS Programmer's Guide

Network Interfaces Programmer's Guide

IEEE 802.3ae Specification - 2002

Name vt – Solaris virtual console interface

Synopsis

```
#include <sys/kd.h>

#include <sys/vt.h>
```

Description The virtual console device driver — also known as virtual terminal (`VT`) — is a layer of management functions that provides facilities to support and switch between multiple screen faces on a single physical device.

VT's are accessed in the same way as other devices. The `open(2)` system call is used to open the virtual console and `read(2)`, `write(2)` and `ioctl(2)` are used in the normal way and support the functionality of the underlying device. In addition, some virtual console-specific ioctls are provided and described below.

The VT provides a link between different screen faces and the device. The *active virtual console* corresponds to the currently visible screen face. Device input is directed to the active console and any device-specific modes that change on a per virtual terminal basis are set to the characteristics associated with the active console.

You manage VT's by intercepting keyboard sequences (“hot key”). To maintain consistency with Xserver, the virtual console device driver supports the Ctrl, Alt, F# and arrow keys.

The sequence **`AltL + F#`** (where AltL represents the Alt key and F# represents function keys 1 through 12) is used to select virtual console 1-12. The sequence **`AltGraph + F#`** (where AltGraph represents the right Alt key and F# represent function keys 1 through 12) is for virtual console 13-24. **`Alt + F1`** chooses the system console (also known as virtual console 1). The sequence **`Alt + ->`** (where “>” represents the right directional arrow) selects the next VT in a circular ring fashion and **`Alt + <-`** (where “<” represents the left directional arrow) changes to the previous console in a circular fashion. The sequence **`Alt + ^`** (where “^” represents the up directional arrow) is for the last used console.

Virtual console switching can be done automatically (`VT_AUTO`) on receipt of a “hot-key" or by the process owning the VT (`VT_PROCESS`). When performed automatically, the process associated with the virtual console is unaware of the switch. Saving and restoring the device are handled by the underlying device driver and the virtual console manager. Note that automatic switching is the default mode.

When a “hot-key" is sent when in process-controlled switch mode, the process owning the VT is sent a signal (relsig) it has specified to the virtual console manager (see `signal(3C)`) requesting the process to release the physical device. At this point, the virtual console manager awaits the `VT_RELDISP` ioctl from the process. If the process refuses to release the device (meaning the switch does not occur), it performs a `VT_RELDISP` ioctl with an argument of 0 (zero). If the process desires to release the device, it saves the device state (keyboard, display, and I/O registers) and then performs a `VT_RELDISP` with an argument of 1 to complete the switch.

A ring of VT's can contain intermixed auto mode and process control mode consoles. When an auto mode process becomes active, the underlying device driver and the virtual console manager handle the restoring of the device. Process control mode processes are sent a specified signal (acqsig) when they become the active console. The process then restores the device state (keyboard, display, and I/O registers) and performs `VT_RELDISP` ioctl with an argument of `VT_ACKACQ` to complete the switching protocol.

The modify-operations ioctls (`VT_SETMODE`, `VT_RELDISP`, `VT_WAITACTIVE`, `KDSETMODE`) check if the VT is the controlling tty of the calling process. If not, the sys_devices privilege is enforced. `VT_ACTIVATE` requires the sys_devices privilege. Note that there is no controlling tty and privilege check for query/view operations.

ioctls The following ioctls apply to devices that support virtual consoles:

`VT_ENABLED`
Queries to determine if VT functionality is available on the system. The argument is a pointer to an integer. If VT functionality is available, the integer is 1, otherwise it is 0.

`VT_OPENQRY`
Finds an available VT. The argument is a pointer to an integer. The integer is filled in with the number of the first available console that no other process has open (and hence, is available to be opened). If there are no available VT's, -1 is filled in.

`VT_GETMODE`
Determines the VT's current mode, either `VT_AUTO` or `VT_PROCESS`. The argument is the address of the following structure, as defined in `<sys/vt.h>`

```
struct vt_mode {
          char mode;  /* VT    mode */
          char waitv; /* not used */
          short relsig;/* signal to use for release request */
          short acqsig;/* signal to use for display acquired */
          short frsig;/* not used */
        }

        /* Virtual console Modes */
        #define    VT_AUTO        0 /* automatic VT switching    */
        #define    VT_PROCESS     1 /* process controls switching */


        The structure will be filled in with the current value
        for each field.
```

`VT_SETMODE`
Sets the VT mode. The argument is a pointer to a vt_mode structure as defined above. The structure should be filled in with the desired mode. If process-control mode is specified, the signals used to communicate with the process should be specified. If any signals are not specified (value is zero), the signal default is `SIGUSR1` (for relsig and acqsig).

VT_RELDISP
: Tells the VT manager if the process releases (or refuses to release) the display. An argument of 1 indicates the VT is released. An argument of 0 indicates refusal to release. The VT_ACKACQ argument indicates if acquisition of the VT has been completed.

VT_ACTIVATE
: Makes the VT specified in the argument the active VT (in the same manner as if a hotkey initiated the switch). If the specified VT is not open or does not exist, the call fails and errno is set to ENXIO.

VT_WAITACTIVE
: If the specified VT is currently active, this call returns immediately. Otherwise, it sleeps until the specified VT becomes active, at which point it returns.

VT_GETSTATE
: Obtains the active VT number and a list of open VTs. The argument is an address to the following structure:

```
struct vt_stat {
        unsigned short    v_active, /* number of the active VT */
                      v_signal, /* not used */
                      v_state;  /* count of open VTs.  For every 1 in this
                                   field, there is an open VT */
        }
```

With VT_GETSTATE, the VT manager first gets the number of the active VT, then determines the number of open VTs in the system and sets a 1 for each open VT in v_state. Next, the VT manager transfers the information in structure vt_stat passed by the user process.

KDGETMODE
: Obtains the text/graphics mode associated with the VT.

```
#define KD_TEXT         0
#define KD_GRAPHICS     1
```

KDSETMODE
: Sets the text/graphics mode to the VT.

KD_TEXT indicates that console text is displayed on the screen. Normally KD_TEXT is combined with VT_AUTO mode for text console terminals, so that the console text display automatically is saved and restored on the hot key screen switches.

KD_GRAPHICS indicates that the user/application (usually Xserver) has direct control of the display for this VT in graphics mode. Normally KD_GRAPHICS is combined with VT_PROCESS mode for this VT indicating direct control of the display in graphics mode. In this mode, all writes to the VT using the write system call are ignored, and you must save and restore the display on the hot key screen switches.

When the mode of the active VT is changed from KD_TEXT to KD_GRAPHICS or a VT of KD_GRAPHICS mode is made active from a previous active VT of KD_TEXT mode, the virtual console manager initiates a KDSETMODE ioctl with KD_GRAPHICS as the argument to the underlying console frame buffer device indicating that current display is running into graphics mode.

When the mode of the active VT is changed from KD_GRAPHICS to KD_TEXT or a VT of KD_TEXT mode is actived from a previous active VT of KD_GRAPHICS mode, the virtual console manager initiates a KDSETMODE ioctl with KD_TEXT as the argument to the underlying console frame buffer device indicating that current display is running into console text mode.

Files /dev/vt/# VT devices.

See Also ioctl(2), signal(3C), wscons(7D)

Notes By default, there are only five virtual console instance login prompts running on /dev/vt/# (where "#" represents 2 to 6) in addition to the system console running on /dev/console. Normally Xorg uses the seventh virtual console (/dev/vt/7.) To switch from consoles to Xserver (which normally picks up the first available virtual console), use [Ctrl +] Alt + F7 .

```
# svcs  | grep login
online         17:49:11 svc:/system/console-login:default
online         17:49:11 svc:/system/console-login:vt2
online         17:49:11 svc:/system/console-login:vt3
online         17:49:11 svc:/system/console-login:vt4
online         17:49:11 svc:/system/console-login:vt5
online         17:49:11 svc:/system/console-login:vt6

console-login:default is for the system console, others for
virtual consoles.

You can modify properties/disable/enable and remove/add
virtual consoles using smf(5):

# svccfg -s console-login add vt8
# svccfg -s console-login:vt8 setprop ttymon/device=astring: "/dev/vt/8"
# svcadm enable console-login:vt8
```

Name vuidmice, vuidm3p, vuidm4p, vuidm5p, vuid2ps2, vuid3ps2 – converts mouse protocol to Firm Events

Synopsis

```
#include <sys/stream.h>

#include <sys/vuid_event.h>

#include <sys/vuid_wheel.h>

int ioctl(fd, I_PUSH, vuidm3p);

int ioctl(fd, I_PUSH, vuidm4p);

int ioctl(fd, I_PUSH, vuidm5p);

int ioctl(fd, I_PUSH, vuid2ps2);

int ioctl(fd, I_PUSH, vuid3ps2);
```

Description The STREAMS modules `vuidm3p`, `vuidm4p`, `vuidm5p`, `vuid2ps2`, and `vuid3ps2` convert mouse protocols to Firm events. The Firm event structure is described in `<sys/vuid_event.h>`. Pushing a STREAMS module does not automatically enable mouse protocol conversion to Firm events. The STREAMS module state is initially set to raw or `VUID_NATIVE` mode which performs no message processing. You must change the state to `VUID_FIRM_EVENT` mode to initiate mouse protocol conversion to Firm events. This can be accomplished by the following code:

```
int format;
format = VUID_FIRM_EVENT;
ioctl(fd, VUIDSFORMAT, &format);
```

You can also query the state of the STREAMS module by using the `VUIDGFORMAT` option.

```
int format;
int fd;    /* file descriptor */
ioctl(fd, VUIDGFORMAT, &format);
if ( format == VUID_NATIVE );
    /* The state of the module is in raw mode.
     * Message processing is not enabled.
     */
if ( format == VUID_FIRM_EVENT );
    /* Message processing is enabled.
     * Mouse protocol conversion to Firm events
     * are performed.
```

The remainder of this section describes the processing of STREAMS messages on the read- and write-side.

Read Side Behavior

`M_DATA` Incoming messages are queued and converted to Firm events.

`M_FLUSH` The read queue of the module is flushed of all its data messages and all data in the record being accumulated are also flushed. The message is passed upstream.

Write Side Behavior

`M_IOCTL` Messages sent downstream as a result of an `ioctl(2)` system call. The two valid `ioctl` options processed by the `vuidmice` modules are `VUIDGFORMAT` and `VUIDSFORMAT`.

`M_FLUSH` The write queue of the module is flushed of all its data messages and the message is passed downstream.

`VUIDGFORMAT` This option returns the current state of the STREAMS module. The state of the `vuidmice` STREAMS module may either be `VUID_NATIVE` (no message processing) or `VUID_FIRM_EVENT` (convert to Firm events).

`VUIDSFORMAT` This option sets the state of the STREAMS module to `VUID_FIRM_EVENT`. If the state of the STREAMS module is already in `VUID_FIRM_EVENT`, this option is non-operational. It is not possible to set the state back to `VUID_NATIVE` once the state becomes `VUID_FIRM_EVENT`. To disable message processing, pop the STREAMS module out by calling `ioctl(fd, 1I_POP, vuid*)`.

The following wheel support ioctls are defined for PS/2 mouse only:

VUIDGWHEELCOUNT This ioctl takes a pointer to an integer as argument and sets the value of the integer to the number of wheels available on this device.

VUIDGWHEELINFO This command returns static information about the wheel that does not change while a device is in use. Currently the only information defined is the wheel orientation which is either VUID_WHEEL_FORMAT_VERTICAL or VUID_WHEEL_FORMAT_HORIZONTAL.

```
typedef struct {
        int     vers;
        int     id;
        int     format;
} wheel_info;
```

The ioctl takes a pointer to "wheel_info" structure with the "vers" set to the current version of the "wheel_info" structure and "id" set to the id of the wheel for which the information is desired.

VUIDSWHEELSTATE
VUIDGWHEELSTATE VUIDSWHEELSTATE sets the state of the wheel to that specified in the stateflags. VUIDGWHEELSTATE returns the current state settings in the stateflags field.

stateflags is an OR'ed set of flag bits. The only flag currently defined is VUID_WHEEL_STATE_ENABLED.

When stateflags is set to VUID_WHEEL_STATE_ENABLED the module converts motion of the specified wheel into VUID events and sends those up stream.

Wheel events are disabled by default.

Applications that want to change a flag should first get the current flags and then change only the bit they want.

```
typedef struct {
        int             vers;
        int             id;
        uint32_t        stateflags;
 } wheel_state;
```

These ioctls take pointer to 'wheel_state' as an argument with the 'vers' and 'id' members filled up. These members have the same meaning as that for 'VUIDGWHEELINFO' ioctl.

Mouse Configurations

Module	Protocol Type	Device
vuidm3p	3-Byte Protocol Microsoft 2 Button Serial Mouse	/dev/tty*
vuidm4p	4-Byte Protocol Logitech 3 Button Mouseman	/dev/tty*
vuidm5p	Logitech 3 Button Bus Mouse Microsoft Bus Mouse	/dev/logi/ dev/msm
vuid2ps2	PS/2 Protocol 2 Button PS/2 Compatible Mouse	/dev/kdmouse
vuid3ps2	PS/2 Protocol 3 Button PS/2 Compatible Mouse	/dev/kdmouse

Attributes

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86

See Also

attributes(5), virtualkm(7D)

STREAMS Programming Guide

Name wpi – Intel Pro Wireless 802.11a/b/g 3945 driver

Description The `wpi` *802.11b/g* wireless NIC driver is a multi-threaded, loadable, clonable, GLDv3-based STREAMS driver supporting the Intel Pro Wireless 3945ABG chipset-based NIC's.

Configuration The `wpi` driver performs auto-negotiation to determine the data rate and mode. Supported *802.11b* data rates are 1, 2, 5.5 and 11 Mbits/sec. Supported *802.11g* data rates are 1, 2, 5.5, 11, 6, 9, 12, 18, 24, 36, 48 and 54 Mbits/sec.

Files

`/dev/wpi`
Special character device.

`/kernel/drv/wpi`
32–bit ELF kernel module (x86).

`/kernel/drv/amd64/wpi`
64–bit ELF kernel module (x86).

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWwpi
Interface Stability	Committed

See Also `dladm(1M)`, `wificonfig(1M)`, `attributes(5)`, `gld(7D)`, `dlpi(7P)`

802.11 — Wireless LAN Media Access Control and Physical Layer Specification — IEEE, 2001

Name wscons – workstation console

Synopsis

```
#include <sys/strredir.h>

ioctl(fd, SRIOCSREDIR, target);

ioctl(fd, SRIOCISREDIR, target);
```

Description The `wscons` workstation console consists of a workstation keyboard and frame buffer that act together to emulate an ASCII terminal. It includes a redirection facility that allows I/O issued to the workstation console to be diverted to a STREAMS device, enabling window systems to redirect output that would otherwise appear directly on the frame buffer in corrupted form.

Redirection The `wscons` redirection facility maintains a list of devices that are designated as redirection targets through the `SRIOCSREDIR` ioctl described below. Only the current entry is active; when the active entry is closed, the most recent remaining entry becomes active. The active entry acts as a proxy for the device being redirected and handles all `read(2)`, `write(2)`, `ioctl(2)`, and `poll(2)` calls issued against the redirectee.

The ioctls described below control the redirection facility. In both cases, *fd* is a descriptor for the device being redirected (or workstation console) and *target* is a descriptor for a STREAMS device.

`SRIOCSREDIR` Designates *target* as the source and destination of I/O ostensibly directed to the device denoted by *fd*.

`SRIOCISREDIR` Returns 1 if *target* names the device currently acting as proxy for the device denoted by *fd*, and 0 if it is not.

ANSI Standard Terminal Emulation The Solaris kernel terminal emulator provides ANSI X3.64 emulation both on SPARC and x86 systems.

On SPARC systems, the PROM monitor is used to emulate an ANSI X3.64 terminal if the kernel terminal emulator is not available for emulation. See `visual_io(7I)` for more details.

Note: The VT100 adheres the ANSI X3.64 standard. However, because the VT100 features nonstandard extensions to ANSI X3.64, it is incompatible with Sun terminal emulators.

The SPARC console displays 34 lines of 80 ASCII characters per line. The x86 console displays 25 lines of 80 ASCII characters per line. Devices with smaller text capacities may display less. On SPARC systems, the `screen-#rows` `screen-#columns` should be set to 34 or 80 respectively or text capacities will vary from those described above. On SPARC systems, the `screen-#rows` and `screen-#columns` fields are stored in NVRAM/EEPROM. See `eeprom(1M)` for more information. Both SPARC and x86 consoles offer scrolling, (x, y) cursor addressing ability and a number of other control functions.

The console cursor marks the current line and character position on the screen. ASCII characters between `0x20` (space) and `0x7E` (tilde) inclusive are printing characters. When a

print character is written to the console (and is not part of an escape sequence), it is displayed at the current cursor position and the cursor moves one position to the right on the current line.

On SPARC based systems, later PROM revisions have the full 8-bit ISO Latin-1 (*ISO 8859-1*) character set. Earlier PROM revisions display characters in the range `0xA0` through `0xFE` as spaces.

When the cursor is at the right edge of the screen, it moves to the first character position on the next line. When the cursor is at the screen's right-bottom edge, the line-feed function is performed (see CTRL-J below). The line-feed function scrolls the screen up by one or more lines before moving the cursor to the first character position on the next line.

Control Sequence Syntax

The `wscons` console defines a number of control sequences that may occur during input. When a control sequence is written to the console, it affects one of the control functions described below. Control sequences are not displayed on screen.

A number of control sequences (or control character functions) are of the form:

`CTRL-`*x*

where *x* represents a singe character., such as `CNTRL-J` for a line feed.

Other ANSI control sequences are of the form:

`ESC [` *params char*

Note – Spaces are included only for readability; these characters must occur in the given sequence without the intervening spaces.

ESC	ASCII escape character (`ESC, CTRL-[, 0x1B`).
[	Left square bracket '[' (`0x5B`).
params	Sequence of zero or more decimal numbers made up of digits between 0 and 9, separated by semicolons. Parameters are represented by *n* in the syntax descriptions for escape sequence functions.
char	Function character, which is different for each control sequence and it represented by *x* in the syntax descriptions for control character functions.

In the following examples of syntactically valid escape sequences, `ESC` represent the single ASCII character, Escape:

ESC[m	Select graphic rendition with default parameter
ESC[7m	Select graphic rendition with reverse image
ESC[33;54H	Set cursor position

ESC[123;456;0;;3;B Move cursor down

Syntactically valid control characters and ANSI escape sequences that are not currently interpreted by the console are ignored.

Each control function requires a specified number of parameters. If fewer parameters are supplied, the remaining parameters (with certain exceptions noted below) default to 1. If more parameters are supplied, the first *n* parameters are used by kernel terminal emulator. In contrast, only the last *n* parameters are used by PROM based emulator, where *n* is the number required by that particular command character.

Parameters which are omitted or set to 0 are reset to the default value of 1 (with certain exceptions). For example, the command character `M` requires one parameter. `ESC[;M`, `ESC[0M`, `ESC[M` and `ESC[23;15;32;1M` are all equivalent to `ESC[1M` and provide a parameter value of 1. Note that `ESC[;5M` (interpreted as '`ESC[5M`') is *not* equivalent to `ESC[5;M` (interpreted as '`ESC[5;1M`') which is ultimately interpreted as '`ESC[1M`').

ANSI Control Functions

The following paragraphs specify the ANSI control functions implemented by the console. Each description provides:

- Control sequence syntax
- Hexadecimal equivalent of control characters where applicable
- Control function name and ANSI or Sun abbreviation (if any).
- Description of parameters required, if any
- Description of the control function
- Initial setting of the mode for functions that set a mode. To restore the initial settings, use the SUNRESET escape sequence.

Control Character Functions

The `wscons` control character functions are:

Bell (BEL),
CTRL-G
0x7 — Used for consoles that are not equipped with an audible bell. Current Sun workstation models also flash the screen if the keyboard is not the console input device.

Backspace (BS),
CTRL-H,
0x8 — The cursor moves one position to the left on the current line. If it is already at the left edge of the screen, no change takes place.

Tab (TAB),
CTRL-I,
0x9 — The cursor moves right on the current line to the next tab stop. The tab stops are fixed at every multiple of eight columns. If the cursor is

already at the right edge of the screen, nothing change takes place. Otherwise, the cursor moves right a minimum of one and a maximum of eight character positions.

Line-feed (LF), CTRL-J, 0xA

The cursor, while remaining at the same character position on the line, moves down one line. If the cursor is at the bottom line, the screen either scrolls up or wraps around depending on the setting of an internal variable *n* (initially 1) . The internal variable can be changed using the `ESC[r` control sequence. If *n* is greater than zero, the entire screen (including the cursor) is scrolled up by *n* lines before executing the line-feed. The top *n* lines scroll off the screen and are lost. New blank lines *n* scroll onto the bottom of the screen. After scrolling, move the cursor down one line to execute the line feed.

If *n* is zero, wrap-around mode is entered. The `ESC [ 1 r` exits back to scroll mode. If a line-feed occurs on the bottom line in wrap mode, the cursor goes to the same character position in the top line of the screen. During line-feeds, the line that the cursor moves to is cleared and no scrolling occurs. Wrap-around mode is not implemented in the window system.

On SPARC based systems, the speed at which the screen scrolls is dependent on the amount of data waiting to be printed. Whenever a scroll occurs and the console is in normal scroll mode (`ESC [ 1 r`), it scans the rest of the data awaiting printing to see how many line-feeds occur in it. This scan stops when the console finds a control character from the set {`VT`, `FF`, `SO`, `SI`, `DLE`, `DC1`, `DC2`, `DC3`, `DC4`, `NAK`, `SYN`, `ETB`, `CAN`, `EM`, `SUB`, `ESC`, `FS`, `GS`, `RS`, `US`} . At that point, the screen is scrolled by *n* lines ($n \geq 1$) and processing continues. The scanned text is processed normally and fills in the newly created lines. As long as escape codes or other control characters are not intermixed with the text, this results in faster scrolling

Reverse Line-feed, CTRL-K, 0xB

With kernel terminal emulator (while remaining at the same character position on the line), the cursor moves down one line. However, with PROM based emulator (while remaining at the same character position on the line), the cursor moves up one line. If the cursor is already at the top line, no change takes place.

Form-feed (FF) CTRL-L,

0xC	The cursor is positioned to the home position (upper-left corner) and the entire screen is cleared.
Return (CR), CTRL-M, 0xD	The cursor moves to the leftmost character position on the current line.

Escape Sequence Functions

The `wscons` escape sequence functions are:

Escape (ESC), CTRL-[, 0x1B	The escape character. Escape initiates a multi-character control sequence.
Insert Character (ICH) ESC[#@	Takes one parameter, *n* (default 1). Inserts *n* spaces at the current cursor position. The current line, starting at the current cursor position inclusive, is shifted to the right by *n* character positions to make room for the spaces. The rightmost *n* character positions shift off the line and are lost. The position of the cursor is unchanged.
Cursor Up (CUU), ESC[#A	Takes one parameter, *n* (default 1). Moves the cursor up *n* lines. If the cursor is fewer than *n* lines from the top of the screen, moves the cursor to the topmost line on the screen. The character position of the cursor on the line is unchanged.
Cursor Down (CUD), ESC[#B	Takes one parameter, (default 1). Moves the cursor down *n* lines. If the cursor is fewer than *n* lines from the bottom of the screen, move the cursor to the last line on the screen. The character position of the cursor on the line is unchanged.
Cursor Forward (CUF), ESC[#C	Takes one parameter, *n* (default 1). Moves the cursor to the right by *n* character positions on the current line. If the cursor is fewer than *n*

	positions from the right edge of the screen, moves the cursor to the rightmost position on the current line.
Cursor Backward (CUB), ESC[#D	Takes one parameter, *n* (default 1). Moves the cursor to the left by *n* character positions on the current line. If the cursor is fewer than *n* positions from the left edge of the screen, moves the cursor to the leftmost position on the current line.
Cursor Next Line (CNL), ESC[#E	Takes one parameter, *n* (default 1). Positions the cursor at the leftmost character position on the *n*-th line below the current line. If the current line is less than *n* lines from the bottom of the screen, positions the cursor at the leftmost character position on the bottom line.
Horizontal and Vertical Position (HVP), ESC[#1;#2f	or
Cursor Position (CUP), ESC[#1;#2H	Takes two parameters, *n*1 and *n*2 (default 1, 1). Moves the cursor to the *n*2-th character position on the *n*1-th line. Character positions are numbered from 1 at the left edge of the screen; line positions are numbered from 1 at the top of the screen. Hence, if both parameters are omitted, the default action moves the cursor to the home position (upper left corner). If only one parameter is supplied, the cursor moves to column 1 of the specified line.
Erase in Display (ED), ESC[J	Takes no parameters. Erases from the current cursor position inclusive to the end of the screen, that is, to the end of the current line and all lines below the current line. The cursor position is unchanged.
Erase in Line (EL), ESC[K	Takes no parameters. Erases from the current cursor position inclusive to the end of the current line. The cursor position is unchanged.

Insert Line (IL),
ESC[#L

Takes one parameter, *n* (default 1). Makes room for *n* new lines starting at the current line by scrolling down by *n* lines the portion of the screen from the current line inclusive to the bottom. The *n* new lines at the cursor are filled with spaces; the bottom *n* lines shift off the bottom of the screen and are lost. The position of the cursor on the screen is unchanged.

Delete Line (DL),
ESC[#M

Takes one parameter, *n* (default 1). Deletes *n* lines beginning with the current line. The portion of the screen from the current line inclusive to the bottom is scrolled upward by *n* lines. The *n* new lines scrolling onto the bottom of the screen are filled with spaces; the *n* old lines beginning at the cursor line are deleted. The position of the cursor on the screen is unchanged.

Delete Character (DCH),
ESC[#P

Takes one parameter, *n* (default 1). Deletes *n* characters starting with the current cursor position. Shifts the tail of the current line to the left by *n* character positions from the current cursor position, inclusive, to the end of the line. Blanks are shifted into the rightmost *n* character positions. The position of the cursor on the screen is unchanged.

Select Graphic Rendition (SGR),
ESC[#m

Takes one parameter, *n* (default 0). Note that unlike most escape sequences, the parameter defaults to zero if omitted. Invokes the graphic rendition specified by the parameter. All following printing characters in the data stream are rendered according to the parameter until the next occurrence of this escape sequence in the data stream. With PROM-based emulator, only two graphic renditions are defined:

0 Normal rendition

7 Negative (reverse) image

Negative image displays characters as white-on-black if the screen mode is currently black-on white, and vice-versa. Any non-zero value of *n* is currently equivalent to 7 and selects the negative image rendition.

In addition to the two renditions mentioned above, the following *ISO 6429-1983* graphic rendition values support color text with kernel terminal emulator:

30 black foreground

31 red foreground

32 green foreground

33 brown foreground

34 blue foreground

35 magenta foreground

36 cyan foreground

37 white foreground

40 black background

41 red background

42 green background

43 brown background

44 blue background

45 magenta background

46 cyan background

47 white background

Black On White (SUNBOW), ESC[p

Takes no parameters. On SPARC, sets the screen mode to black-on-white. If the screen mode is already black-on-white, has no effect. In this mode, spaces display as solid white, other characters as black-on-white. The cursor is a solid black block. Characters displayed in negative image rendition (see 'Select Graphic Rendition' above) are white-on-black. This

	comprises the initial setting of the screen mode on reset. On x86 systems, use `ESC[q` to set black-on-white.
White On Black (SUNWOB), ESC[q	Takes no parameters. On SPARC, sets the screen mode to white-on-black. If the screen mode is already white-on-black, has no effect. In this mode spaces display as solid black, other characters as white-on-black. The cursor is a solid white block. Characters displayed in negative image rendition (see 'Select Graphic Rendition' above) are black-on-white. Initial setting of the screen mode on reset is black on white. On x86 systems, use `ESC[p` to set white-on-black.
ESC[#r Set Scrolling (SUNSCRL)	Takes one parameter, *n* (default 0). Sets to *n* an internal register which determines how many lines the screen scrolls up when a line-feed function is performed with the cursor on the bottom line. A parameter of 2 or 3 introduces a small amount of jump when a scroll occurs. A parameter of 34 clears the screen rather than scrolling. The initial setting is 1 on reset. A parameter of zero initiates wrap mode instead of scrolling. If a linefeed occurs on the bottom line during wrap mode, the cursor goes to the same character position in the top line of the screen. When a line feed occurs, the line that the cursor moves to is cleared and no scrolling occurs. `ESC [ 1 r` exits back to scroll mode. For more information, see the description of the Line-feed (`CTRL-J`) control function above.
ESC[s Reset terminal emulator (SUNRESET)	Takes no parameters. Resets all modes to default, restores current font from PROM. Screen and cursor position are unchanged.

Return Values When there are no errors, the redirection ioctls have return values as described above. Otherwise, they return −1 and set `errno` to indicate the error. If the target stream is in an error state, `errno` is set accordingly.

If the *target* stream is in an error state, `errno` is set accordingly.

Errors

`EBADF` *target* does not denote an open file.

`ENOSTR` *target* does not denote a STREAMS device.

Files

`/dev/wscons` Workstation console, accessed via the redirection facility

`/dev/systty` Devices that must be opened for the `SRIOCSREDIR` and `SRIOCISREDIR` ioctls.

`/dev/syscon` Access system console

`/dev/console` Access system console

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Stable

See Also

`cvcd(1M)`, `eeprom(1M)`, `ioctl(2)`, `poll(2)`, `read(2)`, `write(2)`, `cvc(7D)`, `console(7D)`, `visual_io(7I)`

Warnings

The redirection ioctls block while there is I/O outstanding on the device instance being redirected. If you try to redirect the workstation console while there is a outstanding read, the workstation console will hang until the read completes.

Notes

The `cvc` facility supersedes the SunOS `wscons` facility and should not be used with `wscons`.

Name wusb_ca – WUSB Cable Association Driver

Description The `wusb_ca` driver is a USBA (Solaris USB Architecture) compliant client driver that supports the cable association model which is defined in Association Models Supplement to the Certified WUSB specification.

The wireless USB cable association driver is a USB class driver that provides interfaces for establishing a first-time connection between Wireless USB hosts and devices. This process of establishing a first-time connection is called *association* in WUSB standard. It is a prerequisite process that must be completed by hosts and devices prior to implementing the security requirements outlined in *Wireless Universal Serial Bus Specification 1.0*.

Users should use `wusbadm`(1M) to do cable association for WUSB devices.

Files

`/kernel/drv/wusb_ca`	32-bit ELF 86 kernel module
`/kernel/drv/amd64/wusb_ca`	64-bit x86 ELF kernel module
`/kernel/drv/sparcv9/wusb_ca`	64-bit SPARC ELF kernel module

Attributes See `attributes`(5) for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86, PCI-based systems
Availability	SUNWusb

See Also `wusbadm`(1M), `attributes`(5), `hwahc`(7D), `hwarc`(7D), `usba`(7D)

Writing Device Drivers

System Administration Guide: Basic Administration

Wireless Universal Serial Bus Specification 1.0

http://www.usb.org

http://www.sun.com

Name wusb_df, hwa1480_fw – WUSB firmware download driver and firmware module for Intel i1480 chipset

Description The `wusb_df` driver is a Solaris USB Architecture (USBA) compliant client driver that is used to download firmware for Host Wire Adapter (HWA) dongles that use Intel i1480 chipsets.

Currently, the `wusb_df` driver can only download driver for Intel i1480 based HWA dongles. The `hwa1480_fw` is a miscellaneous module which is transformed from Intel's firmware binary version RC1.3PA2-20070828. `wusb_df` reads firmware data from `hwa1480_fw` module and downloads it to HWA hardware.

Users can use `elfwrap(1)` to transform new firmware binary. Users must use the same name as `hwa1480_fw`, since `wusb_df` only recognizes this symbol.

Files

`/kernel/drv/wusb_df`	32-bit ELF 86 kernel module
`/kernel/drv/sparcv9/wusb_df`	64-bit SPARC ELF kernel module
`/kernel/drv/amd64/wusb_df`	64-bit x86 ELF kernel module
`/kernel/misc/hwa_1480`	32-bit ELF 86 kernel module
`/kernel/misc/sparcv9/hwa_1480`	64-bit SPARC ELF kernel module
`/kernel/drv/amd64/hwa_1480`	64-bit x86 ELF kernel module

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, x86, PCI-based systems
Availability	SUNWusb

See Also `elfwrap(1)`, `add_drv(1M)`, `rem_drv(1M)`, `update_drv(1M)`, `attributes(5)`

Writing Device Drivers

System Administration Guide: Basic Administration

Wireless Universal Serial Bus Specification 1.0

http://www.usb.org

http://www.sun.com

Name xge – Neterion Xframe 10Gigabit Ethernet Network Adapter driver

Synopsis `/dev/xge`

Description The `xge` 10 Gigabit Ethernet driver is a multi-threaded, loadable, clonable, GLD-based STREAMS driver supporting the Data Link Provider Interface, `dlpi(7P)`, on S2IO Xframe 10-Gigabit Ethernet Network Adapter.

The `xge` driver functions includes controller initialization, frame transmit and receive, promiscuous and multicast support, TCP and UDP checksum offload (IPv4 and IPv6), 9622-byte jumbo frame, and error recovery and reporting.

The `xge` driver and hardware support the 10GBase-SR/W, LR/W, and ER/W *802.3* physical layer.

Application Programming Interface The cloning, character-special device `/dev/xge` is used to access all Xframe devices installed within the system.

The `xge` driver is managed by the `dladm(1M)` command line utility, which allows VLANs to be defined on top of xge instances and for `xge` instances to be aggregated. See `dladm(1M)` for more details.

The values returned by the driver in the DL_INFO_ACK primitive in response to the DL_INFO_REQ are as follows:

- Maximum SDU is `9600`.
- Minimum SDU is `0`.
- DSLAP address length is `8` bytes.
- MAC type is `DL_ETHER`.
- SAP length value is -2 meaning the physical address component is followed immediately by a 2-byte sap component within the DLSAP address.
- Broadcast address value is Ethernet/IEEE broadcast address (FF:FF:FF:FF:FF:FF).

Configuration By default, the `xge` driver works without any configuration file.

You can check the running-time status of a device instance using `ndd(1M)`. Currently, the driver provides an interface to print all hardware statistics.

For example, to print statistics of device xge0:

```
#ndd /dev/xge0 stats
tmac_data_octets 772
tmac_frms 15
tmac_drop_frms 0
tmac_bcst_frms 6
tmac_mcst_frms 6
```

```
...

rmac_vld_frms 13
rmac_fcs_err_frms 0
rmac_drop_frms 0
rmac_vld_bcst_frms 7
rmac_vld_mcst_frms 11
rmac_out_rng_len_err_frms 0
rmac_in_rng_len_err_frms 0
rmac_long_frms 0

...

not_traffic_intr_cnt 242673
traffic_intr_cnt 28

...
```

Files `/dev/xge` xge special character device

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86

See Also `dladm(1M)`, `ndd(1M)`, `attributes(5)`, `streamio(7I)`, `dlpi(7P)`

Writing Device Drivers

STREAMS Programming Guide

Network Interfaces Programmer's Guide

Name yge – Marvell Yukon 2 Ethernet device driver

Synopsis `/dev/net/yge`

Description The yge driver supports Marvell Yukon 2 Fast Ethernet and Gigabit Ethernet controllers.

Properties The following properties can be configured using `dladm(1M)`:

`adv_autoneg_cap`	Enables (default) or disables IEEE 802.3 auto-negotiation of link speed and duplex settings. If enabled, the device negotiates among the supported (and configured, see below) link options with the link partner. If disabled, at least one of the link options below must be specified. The driver selects the first enabled link option according to the IEEE 802.3 specified preferences.
`adv_1000fdx_cap`	Enables the 1000 Mbps full-duplex link option.
`adv_1000hdx_cap`	Enables the 1000 Mbps half-duplex link option.
`adv_100T4_cap`	Enables the 100 BaseT4 link option. (Note that most hardware does not support this unusual link style. Also, this uses two pairs of wires for data, rather than one.)
`adv_100fdx_cap`	Enables the 1000 Mbps full-duplex link option.
`adv_100hdx_cap`	Enables the 1000 Mbps half-duplex link option.
`adv_10fdx_cap`	Enables the 10 Base-T full-duplex link option.
`adv_10hdx_cap`	Enables the 10 Base-T half-duplex link option.
`mtu`	On most devices, can be set between 1500 (default) and 9000. This property can only be changed when the device is not in use.

Files

`/dev/net/yge`	Special network device
`/kernel/drv/yge`	32-bit driver binary (x86)
`/kernel/drv/amd64/yge`	64-bit driver binary (x86)
`/kernel/drv/sparcv9/yge`	64-bit driver binary (SPARC)

Attributes See `attributes(5)` for a descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86, SPARC
Interface Stability	Committed

See Also dladm(1M), attributes(5), dlpi(7P)

Name zcons – Zone console device driver

Description The `zcons` character driver exports the console for system zones. The driver is comprised of two "sides:" a master side with which applications in the global zone communicate, and a slave side, which receives I/O from the master side. The slave side is available in the global zones.

Applications must not depend on the location of `/dev` or `/devices` entries exported by `zcons`. Inside a zone, the `zcons` slave side is fronted by `/dev/console` and other console-related symbolic links, which are used by applications that expect to write to the system console.

The `zcons` driver is Sun Private, and may change in future releases.

Files

`/dev/zcons/<`*zonename*`>/masterconsole`	Global zone master side console for zone <*zonename*>.
`/dev/zcons/<`*zonename*`>/slaveconsole`	Global zone slave side console for zone <*zonename*>.
`/dev/zconsole`	Non-global zone console (slave side).

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
Interface Stability	Sun Private

See Also `zoneadm(1M)`, `zonecfg(1M)`, `attributes(5)`, `zones(5)`

Name zero – source of zeroes

Description A zero special file is a source of zeroed unnamed memory.

Reads from a zero special file always return a buffer full of zeroes. The file is of infinite length.

Writes to a zero special file are always successful, but the data written is ignored.

Mapping a zero special file creates a zero-initialized unnamed memory object of a length equal to the length of the mapping and rounded up to the nearest page size as returned by `sysconf`. Multiple processes can share such a zero special file object provided a common ancestor mapped the object `MAP_SHARED`.

Files `/dev/zero`

See Also `fork`(2), `mmap`(2), `sysconf`(3C)

Name zs – Zilog 8530 SCC serial communications driver

Synopsis

```
#include <fcntl.h>

#include <sys/termios.h>

open("/dev/term/n", mode);

open("/dev/ttyn", mode);

open("/dev/cua/n", mode);
```

Description The Zilog 8530 provides two serial input/output channels capable of supporting a variety of communication protocols. A typical system uses two or more of these devices to implement essential functions, including RS-423 ports (which also support most RS-232 equipment), and the console keyboard and mouse devices.

The `zs` module is a loadable STREAMS driver that provides basic support for the Zilog 8530 hardware and basic asynchronous communication support. The driver supports the `termio(7I)` device control functions specified by flags in the `c_cflag` word of the `termios` structure and by the `IGNBRK`, `IGNPAR`, `PARMRK`, or `INPCK` flags in the `c_iflag` word. All other `termio(7I)` functions must be performed by STREAMS modules pushed atop the driver. When a device is opened, the `ldterm(7M)` and `ttcompat(7M)` STREAMS modules are automatically pushed on top of the stream, providing the standard `termio(7I)` interface.

The character-special devices `/dev/term/a` and `/dev/term/b` are used to access the two serial ports on the CPU board.

Valid name space entries are `/dev/cua/`[*a-z*], `/dev/term/`[*a-z*] and `/dev/tty`[*a-z*]. The number of entries used in a name space are machine dependent.

To allow a single tty line to be connected to a modem and used for both incoming and outgoing calls, a special feature is available that is controlled by the minor device number. By accessing character-special devices with names of the form `/dev/cua/`[*n*], it is possible to open a port without the `Carrier Detect` signal being asserted, either through hardware or an equivalent software mechanism. These devices are commonly known as dial-out lines.

Once a `/dev/cua/`[*n*] line is opened, the corresponding tty line cannot be opened until the `/dev/cua/`*n* line is closed. A blocking open will wait until the `/dev/cua/`[*n*] line is closed (which will drop `Data Terminal Ready`, and `Carrier Detect`) and carrier is detected again. A non-blocking open will return an error. If the tty line has been opened successfully (usually only when carrier is recognized on the modem) , the corresponding `/dev/cua/`[*n*] line cannot be opened. This allows a modem to be attached to `/dev/term/`[*n*] (renamed from `/dev/tty`[*n*]) and used for dial-in (by enabling the line for login in `/etc/inittab`) and also used for dial-out (by `tip(1)` or `uucp(1C)`) as `/dev/cua/`[*n*] when no one is logged in on the line.

Note – This module is affected by the setting of specific eeprom variables. For information on parameters that are persistent across reboots, see the `eeprom(1M)` man page.

ioctls The `zs` module supports the standard set of `termio ioctl( )` calls.

If the `CRTSCTS` flag in the `c_cflag` field is set, output will be generated only if `CTS` is high; if `CTS` is low, output will be frozen. If the `CRTSCTS` flag is clear, the state of CTS has no effect.

If the `CRTSXOFF` flag in the `c_cflag` field is set, input will be received only if RTS is high; if RTS is low, input will be frozen. If the `CRTSXOFF` flag is clear, the state of `RTS` has no effect.

The `termios CRTSCTS` (respectively `CRTSXOFF)` flag and `termiox CTSXON` (respectively `RTSXOFF)` can be used interchangeably.

Breaks can be generated by the `TCSBRK, TIOCSBRK,` and `TIOCCBRK ioctl( )` calls.

The state of the `DCD, CTS, RTS,` and `DTR` interface signals may be queried through the use of the `TIOCM_CAR, TIOCM_CTS, TIOCM_RTS,` and `TIOCM_DTR` arguments to the `TIOCMGET ioctl` command, respectively. Due to hardware limitations, only the `RTS` and `DTR` signals may be set through their respective arguments to the `TIOCMSET, TIOCMBIS,` and `TIOCMBIC ioctl` commands.

The input and output line speeds may be set to any of the speeds supported by `termio`. The input and output line speeds cannot be set independently; for example, when you set the the output speed, the input speed is automatically set to the same speed.

When the driver is used to service the serial console port, it supports a BREAK condition that allows the system to enter the debugger or the monitor. The BREAK condition is generated by hardware and it is usually enabled by default. A BREAK condition originating from erroneous electrical signals cannot be distinguished from one deliberately sent by remote DCE. The Alternate Break sequence can be used to remedy this.

Due to a risk of incorrect sequence interpretation, SLIP and certain other binary protocols should not be run over the serial console port when Alternate Break sequence is in effect. Although PPP is a binary protocol, it is able to avoid these sequences using the ACCM feature in *RFC 1662*. For Solaris PPP 4.0, you do this by adding the following line to the `/etc/ppp/options` file (or other configuration files used for the connection; see `pppd(1M)` for details):

```
asyncmap  0x00002000
```

By default, the Alternate Break sequence is three characters: carriage return, tilde and control-B (CR ~ CTRL-B), but may be changed by the driver. For more information on breaking (entering the debugger or monitor), see `kbd(1)` and `kb(7M)`.

Errors An open will fail under the following conditions:

`ENXIO` The unit being opened does not exist.

`EBUSY` The dial-out device is being opened and the dial-in device is already open, or the dial-in device is being opened with a no-delay open and the dial-out device is already open.

`EBUSY` The port is in use by another serial protocol.

`EBUSY` The unit has been marked as exclusive-use by another process with a `TIOCEXCL` `ioctl( )` call.

`EINTR` The open was interrupted by the delivery of a signal.

Files

`/dev/cua/`[*a-z*] dial-out tty lines

`/dev/term/`[*a-z*] dial-in tty lines

`/dev/tty`[*a-z*] binary compatibility package device names

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC

See Also `eeprom(1M)`, `kmdb(1)`, `tip(1)`, `cu(1C)`, `uucp(1C)`, `ports(1M)`, `pppd(1M)`, `ioctl(2)`, `open(2)`, `attributes(5)`, `zsh(7D)`, `termio(7I)`, `kb(7M)`, `ldterm(7M)`, `ttcompat(7M)`

Diagnostics

`zs`*n* `: silo overflow.` The Zilog 8530 character input silo overflowed before it could be serviced.

`zs`*n* `: ring buffer overflow.` The driver's character input ring buffer overflowed before it could be serviced.

Name zsh – On-board serial HDLC/SDLC interface

Synopsis

```
#include <fcntl.h>

open(/dev/zshn, mode );

open(/dev/zsh, mode );
```

Description The zsh module is a loadable STREAMS driver that implements the sending and receiving of data packets as HDLC frames over synchronous serial lines. The module is not a standalone driver, but instead depends upon the zs module for the hardware support required by all on-board serial devices. When loaded this module acts as an extension to the zs driver, providing access to an HDLC interface through character-special devices.

The zsh*n* devices provide what is known as a data path which supports the transfer of data via read(2) and write(2) system calls, as well as ioctl(2) calls. Data path opens are exclusive in order to protect against injection or diversion of data by another process.

The zsh device provides a separate control path for use by programs that need to configure or monitor a connection independent of any exclusive access restrictions imposed by data path opens. Up to three control paths may be active on a particular serial channel at any one time. Control path accesses are restricted to ioctl(2) calls only; no data transfer is possible.

When used in synchronous modes, the Z8530 SCC supports several options for clock sourcing and data encoding. Both the transmit and receive clock sources can be set to be the external Transmit Clock (TRxC), external Receive Clock (RTxC), the internal Baud Rate Generator (BRG), or the output of the SCC's Digital Phase-Lock Loop (DPLL).

The Baud Rate Generator is a programmable divisor that derives a clock frequency from the PCLK input signal to the SCC. A programmed baud rate is translated into a 16-bit time constant that is stored in the SCC. When using the BRG as a clock source the driver may answer a query of its current speed with a value different from the one specified. This is because baud rates translate into time constants in discrete steps, and reverse translation shows the change. If an exact baud rate is required that cannot be obtained with the BRG, an external clock source must be selected.

Use of the DPLL option requires the selection of NRZI data encoding and the setting of a non-zero value for the baud rate, because the DPLL uses the BRG as its reference clock source.

A local loopback mode is available, primarily for use by the syncloop(1M) utility for testing purposes, and should not be confused with SDLC loop mode, which is not supported on this interface. Also, an auto-echo feature may be selected that causes all incoming data to be routed to the transmit data line, allowing the port to act as the remote end of a digital loop. Neither of these options should be selected casually, or left in use when not needed.

The zsh driver keeps running totals of various hardware generated events for each channel. These include numbers of packets and characters sent and received, abort conditions detected

by the receiver, receive CRC errors, transmit underruns, receive overruns, input errors and output errors, and message block allocation failures. Input errors are logged whenever an incoming message must be discarded, such as when an abort or CRC error is detected, a receive overrun occurs, or when no message block is available to store incoming data. Output errors are logged when the data must be discarded due to underruns, CTS drops during transmission, CTS timeouts, or excessive watchdog timeouts caused by a cable break.

ioctls The zsh driver supports several ioctl() commands, including:

S_IOCGETMODE — Return a struct scc_mode containing parameters currently in use. These include the transmit and receive clock sources, boolean loopback and NRZI mode flags and the integer baud rate.

S_IOCSETMODE — The argument is a struct scc_mode from which the SCC channel will be programmed.

S_IOCGETSTATS — Return a struct sl_stats containing the current totals of hardware-generated events. These include numbers of packets and characters sent and received by the driver, aborts and CRC errors detected, transmit underruns, and receive overruns.

S_IOCCLRSTATS — Clear the hardware statistics for this channel.

S_IOCGETSPEED — Returns the currently set baud rate as an integer. This may not reflect the actual data transfer rate if external clocks are used.

S_IOCGETMCTL — Returns the current state of the CTS and DCD incoming modem interface signals as an integer.

The following structures are used with zsh ioctl() commands:

```
struct  scc_mode {
    char  sm_txclock;   /* transmit clock sources */
    char  sm_rxclock;   /* receive clock sources */
    char  sm_iflags;    /* data and clock inversion flags (non-zsh) */
    uchar_t  sm_config; /* boolean configuration options */
    int  sm_baudrate;   /* real baud rate */
    int  sm_retval;     /* reason codes for ioctl failures */
};
struct  sl_stats {
    long  ipack;        /* input packets */
    long  opack;        /* output packets */
    long  ichar;        /* input bytes */
    long  ochar;        /* output bytes */
    long  abort;        /* abort received */
    long  crc;          /* CRC error */
    long  cts;          /* CTS timeouts */
    long  dcd;          /* Carrier drops */
    long  overrun;      /* receive overrun */
```

```
    long  underrun;     /* transmit underrun */
    long  ierror;       /* input error */
    long  oerror;       /* output error */
    long  nobuffers;    /* receive side memory allocation failure */
};
```

Errors An `open()` will fail if a `STREAMS` message block cannot be allocated, or:

`ENXIO` The unit being opened does not exist.

`EBUSY` The device is in use by another serial protocol.

An `ioctl()` will fail if:

`EINVAL` An attempt was made to select an invalid clocking source.

`EINVAL` The baud rate specified for use with the baud rate generator would translate to a null time constant in the `SCC`'s registers.

Files

`/dev/zsh[0-1],/dev/zsh`	character-special devices
`/usr/include/sys/ser_sync.h`	header file specifying synchronous serial communication definitions

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86

See Also `syncinit(1M)`, `syncloop(1M)`, `syncstat(1M)`, `ioctl(2)`, `open(2)`, `read(2)`, `write(2)`, `attributes(5)`, `zs(7D)`

Refer to the *Zilog Z8530 SCC Serial Communications Controller Technical Manual* for details of the `SCC`'s operation and capabilities.

Diagnostics `zsh data open failed, no memory, rq=`*nnn*

`zsh clone open failed, no memory, rq=`*nnn*
A kernel memory allocation failed for one of the private data structures. The value of *nnn* is the address of the read queue passed to `open(2)`.

`zsh_open: can't alloc message block`
The open could not proceed because an initial STREAMS message block could not be made available for incoming data.

`zsh: clone device` *d* `must be attached before use!`
An operation was attempted through a control path before that path had been attached to a particular serial channel.

`zsh`*n*`: invalid operation for clone dev.`
An inappropriate STREAMS message type was passed through a control path. Only `M_IOCTL` and `M_PROTO` message types are permitted.

`zsh`*n*`: not initialized, can't send message`
An `M_DATA` message was passed to the driver for a channel that had not been programmed at least once since the driver was loaded. The `SCC`'s registers were in an unknown state. The `S_IOCSETMODE` ioctl command performs the programming operation.

`zsh`*n*`: transmit hung`
The transmitter was not successfully restarted after the watchdog timer expired.

Name zulu – Sun XVR-4000 Graphics Accelerator driver

Description The zulu driver is the device driver for the Sun XVR-4000 Graphics Accelerator.

Files

`/dev/fbs/zulu`*n*	device special file
`/usr/lib/zulu.ucode`	`zulu` microcode

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWzulux

See Also `SUNWzulu_config(1M)`, `zuludaemon(1M)`, `attributes(5)`

Name zyd – ZyDAS ZD1211/ZD1211B USB 802.11b/g Wireless Driver

Description The `zyd` *802.11b/g* wireless driver is a multi-threaded, loadable, clonable, GLDv3-based STREAMS driver supporting the ZyDAS ZD1211/ZD1211B USB chipset-based wireless devices.

Configuration The `zyd` driver performs auto-negotiation to determine the data rate and mode. Supported *802.11b* data rates are 1, 2, 5.5 and 11 Mbits/sec. Supported *802.11g* data rates are 1, 2, 5.5, 11, 6, 9, 12, 18, 24, 36, 48 and 54 Mbits/sec. The `zyd` driver supports only BSS networks (also known as "ap" or "infrastructure" networks) and open (or "open-system"), shared key and WPA/WPA2 authentication. Supported encryption types are WEP40, WEP104, TKIP and AES-CCMP.

Files

`/dev/zyd*`	Special character device.
`/kernel/drv/zyd`	32–bit kernel module (x86).
`/kernel/drv/amd64/zyd`	64–bit kernel module (x86).

Attributes See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	x86
Availability	SUNWzyd
Interface Stability	Committed

See Also `dladm(1M)`, `wificonfig(1M)`, `attributes(5)`, `gld(7D)`, `dlpi(7P)`

802.11 - Wireless LAN Media Access Control and Physical Layer Specification - IEEE, 2001