man Pages(7): Device and Network Interfaces
Contents

PREFACE xi
Intro(7) 2
adp(7D) 13
aha(7D) 15
aic(7D) 17
arp(7P) 19
asy(7D) 22
ata(7D) 25
audio(7I) 27
audioamd(7D) 38
audiocs(7D) 40
bd(7M) 43
bpp(7D) 45
bufmod(7M) 51
bwtwo(7D) 56
cdio(7I) 57
cgeight(7D) 68
cgfour(7D) 70
cgfourteen(7D) 71
pcsvi(7D)  328
pcser(7D)  329
pe(7D)    330
pfmod(7M)  332
pipemod(7M)  336
pln(7D)    337
pm(7D)     339
pmc(7D)    343
ppp(7M)    345
ptem(7M)   347
ptm(7D)    348
pts(7D)    350
pty(7D)    352
qe(7D)     356
qec(7D)    359
qfe(7D)    360
quotactl(7I)  365
rns_smt(7D)  368
route(7P)  369
routing(7P)  375
sad(7D)    377
sbpro(7D)  381
sd(7D)     384
se(7D)     391
se_hdlc(7D)  395
ses(7D)    399
sesio(7I)  400
sf(7D)     402
uscsi(7I) 547
visual_io(7I) 552
volfs(7FS) 559
vuidmice(7M) 561
wscons(7D) 564
xd(7D) 574
xt(7D) 576
xy(7D) 578
zero(7D) 580
zs(7D) 581
zsh(7D) 584
Index 588
Overview

A man page is provided for both the naive user, and sophisticated user who is familiar with the SunOS operating system and is in need of on-line information. 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.

The following contains a brief description of each section in the man pages 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 of this volume.
- 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 6 contains available games and demos.
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 9 provides reference information needed to write device drivers in the kernel operating systems environment. It describes two device driver interface specifications: the Device Driver Interface (DDI) and the Driver/Kernel Interface (DKI).

Section 9E describes the DDI/DKI, DDI-only, and DKI-only entry-point routines a developer may 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 pathname 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:

[ ] The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.

... Ellipses. Several values may be provided for the previous argument, or the previous argument can be specified multiple times, for example, \'"filename...\'.

xii man Pages(7):Device and Network Interfaces
♦ October 1998
| Separator. Only one of the arguments separated by this character can be specified at 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, functions and such, are described under USAGE.

IOCTL
This section appears on pages in Section 7 only. Only the device class which 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(7D)

OPTIONS
This 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 SYNOPTIS 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 is provided as a guidance on use. This section lists special rules, features and commands that require in-depth explanations. The subsections listed below 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 filenames 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.
Device and Network Interfaces
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:

- 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)).
This section describes the programmatic interface for several file systems supported by SunOS.

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 \texttt{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.

This section describes STREAMS modules. Note that STREAMS drivers are discussed in section 7D. \texttt{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.

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 \texttt{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 \texttt{socket(3N)} for details.

Major protocols in the Internet family include:

- The Internet Protocol (IP) itself, which supports the universal datagram format, as described in \texttt{ip(7P)}. This is the default protocol for SOCK_RAW type sockets within the AF_INET domain.
- The Transmission Control Protocol (TCP); see \texttt{tcp(7P)}. This is the default protocol for SOCK_STREAM type sockets.
- The User Datagram Protocol (UDP); see \texttt{udp(4P)}. This is the default protocol for SOCK_DGRAM type sockets.
- The Address Resolution Protocol (ARP); see \texttt{arp(7P)}.
- The Internet Control Message Protocol (ICMP); see \texttt{icmp(7P)}.

\textbf{SEE ALSO} \texttt{add_drv(1M)}, \texttt{rem_drv(1M)}, \texttt{intro(2)}, \texttt{ioctl(2)}, \texttt{socket(3N)}, \texttt{driver.conf(4)}, \texttt{arp(7P)}, \texttt{icmp(7P)}, \texttt{inet(7P)}, \texttt{ip(7P)}, \texttt{mtio(7I)}, \texttt{st(7D)}, \texttt{streamio(7I)}, \texttt{tcp(7P)}, \texttt{udp(7P)}
### Solaris 1.x to 2.x Transition Guide

**TCP/IP and Data Communications Administration Guide**

**STREAMS Programming Guide**

**Writing Device Drivers**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARP(7P)</td>
<td>See arp(7P)</td>
</tr>
<tr>
<td>ICMP(7P)</td>
<td>See icmp(7P)</td>
</tr>
<tr>
<td>IP(7P)</td>
<td>See ip(7P)</td>
</tr>
<tr>
<td>Intro(7)</td>
<td>introduction to special files</td>
</tr>
<tr>
<td>TCP(7P)</td>
<td>See tcp(7P)</td>
</tr>
<tr>
<td>UDP(7P)</td>
<td>See udp(7P)</td>
</tr>
<tr>
<td>adp(7D)</td>
<td>low-level module for controllers based on Adaptec AIC-7870P and AIC-7880P SCSI chips</td>
</tr>
<tr>
<td>aha(7D)</td>
<td>low-level module for Adaptec 154x ISA host bus adapters</td>
</tr>
<tr>
<td>aic(7D)</td>
<td>low-level module for Adaptec AIC-6360 based ISA host bus adapters</td>
</tr>
<tr>
<td>arp(7P)</td>
<td>Address Resolution Protocol</td>
</tr>
<tr>
<td>asy(7D)</td>
<td>asynchronous serial port driver</td>
</tr>
<tr>
<td>ata(7D)</td>
<td>AT attachment disk driver</td>
</tr>
<tr>
<td>audio(7I)</td>
<td>generic audio device interface</td>
</tr>
<tr>
<td>audioamd(7D)</td>
<td>telephone quality audio device</td>
</tr>
<tr>
<td>audioscs(7D)</td>
<td>Crystal Semiconductor 4231 audio Interface</td>
</tr>
<tr>
<td>bd(7M)</td>
<td>SunButtons and SunDials STREAMS module</td>
</tr>
<tr>
<td>bpp(7D)</td>
<td>bi-directional parallel port driver</td>
</tr>
<tr>
<td>bufmod(7M)</td>
<td>STREAMS Buffer Module</td>
</tr>
<tr>
<td>btwo(7D)</td>
<td>black and white memory frame buffer</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cdio(7I)</td>
<td>CD-ROM control operations</td>
</tr>
<tr>
<td>cgeight(7D)</td>
<td>24-bit color memory frame buffer</td>
</tr>
<tr>
<td>cgfour(7D)</td>
<td>P4-bus 8-bit color memory frame buffer</td>
</tr>
<tr>
<td>cgfourteen(7D)</td>
<td>4-bit color graphics device</td>
</tr>
<tr>
<td>cgsix(7D)</td>
<td>accelerated 8-bit color frame buffer</td>
</tr>
<tr>
<td>cgthree(7D)</td>
<td>8-bit color memory frame buffer</td>
</tr>
<tr>
<td>cgtwo(7D)</td>
<td>color graphics interface</td>
</tr>
<tr>
<td>cmdk(7D)</td>
<td>common disk driver</td>
</tr>
<tr>
<td>cnft(7D)</td>
<td>device driver for Compaq NIC</td>
</tr>
<tr>
<td>connld(7M)</td>
<td>line discipline for unique stream connections</td>
</tr>
<tr>
<td>console(7D)</td>
<td>STREAMS-based console interface</td>
</tr>
<tr>
<td>cpqncr(7D)</td>
<td>low-level module for Compaq 32-Bit Fast-Wide SCSI-2 EISA/PCI (825) and Compaq Wide-Ultra SCSI PCI (875) Controllers</td>
</tr>
<tr>
<td>cpr(7)</td>
<td>suspend and resume module</td>
</tr>
<tr>
<td>csa(7D)</td>
<td>low-level module for Compaq SMART Array Controller</td>
</tr>
<tr>
<td>cvc(7D)</td>
<td>virtual console driver</td>
</tr>
<tr>
<td>cvcredir(7D)</td>
<td>virtual console redirection driver</td>
</tr>
<tr>
<td>dbri(7D)</td>
<td>Dual Basic Rate ISDN and audio Interface</td>
</tr>
<tr>
<td>devinfo(7D)</td>
<td>device information driver</td>
</tr>
<tr>
<td>display(7D)</td>
<td>system console display</td>
</tr>
<tr>
<td>dkio(7I)</td>
<td>disk control operations</td>
</tr>
<tr>
<td>dlpi(7P)</td>
<td>Data Link Provider Interface</td>
</tr>
<tr>
<td>dnet(7D)</td>
<td>Ethernet driver for DEC 21040, 21041, 21140 Ethernet cards</td>
</tr>
<tr>
<td>dpt(7D)</td>
<td>DPT 2011, 2012, 2021, 2022, 2122, 2124, 2124, 3021, 3222, and 3224 controllers</td>
</tr>
<tr>
<td>Device and Network Interfaces</td>
<td>Intro(7)</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td><strong>ecpp(7D)</strong></td>
<td>IEEE 1284 ecp, nibble and centronics compatible parallel port driver</td>
</tr>
<tr>
<td><strong>eeapro(7D)</strong></td>
<td>Intel EtherExpress-Pro Ethernet device driver</td>
</tr>
<tr>
<td><strong>eha(7D)</strong></td>
<td>low-level module for Adaptec 174x EISA host bus adapter</td>
</tr>
<tr>
<td><strong>elx(7D)</strong></td>
<td>3COM EtherLink III Ethernet device driver</td>
</tr>
<tr>
<td><strong>esa(7D)</strong></td>
<td>low-level module for Adaptec 7770 based SCSI controllers</td>
</tr>
<tr>
<td><strong>esp(7D)</strong></td>
<td>ESP SCSI Host Bus Adapter Driver</td>
</tr>
<tr>
<td><strong>fas(7D)</strong></td>
<td>FAS SCSI Host Bus Adapter Driver</td>
</tr>
<tr>
<td><strong>fbio(7I)</strong></td>
<td>frame buffer control operations</td>
</tr>
<tr>
<td><strong>fd(7D)</strong></td>
<td>drivers for floppy disks and floppy disk controllers</td>
</tr>
<tr>
<td><strong>fd(7D)</strong></td>
<td>See <strong>fd(7D)</strong></td>
</tr>
<tr>
<td><strong>fdio(7I)</strong></td>
<td>floppy disk control operations</td>
</tr>
<tr>
<td><strong>ffb(7D)</strong></td>
<td>24-bit UPA color frame buffer and graphics accelerator</td>
</tr>
<tr>
<td><strong>flashpt(7D)</strong></td>
<td>low-level module for Mylex/BusLogic host bus adapters</td>
</tr>
<tr>
<td><strong>gld(7D)</strong></td>
<td>Generic LAN Driver</td>
</tr>
<tr>
<td><strong>glm(7D)</strong></td>
<td>GLM SCSI Host Bus Adapter Driver</td>
</tr>
<tr>
<td><strong>hdio(7I)</strong></td>
<td>SMD and IPI disk control operations</td>
</tr>
<tr>
<td><strong>hme(7D)</strong></td>
<td>SUNW,hme Fast-Ethernet device driver</td>
</tr>
<tr>
<td><strong>hsfs(7FS)</strong></td>
<td>High Sierra ISO 9660 CD-ROM filesystem</td>
</tr>
<tr>
<td><strong>i2o_bs(7D)</strong></td>
<td>Block Storage OSM for I2O</td>
</tr>
<tr>
<td><strong>i2o_scsci(7D)</strong></td>
<td>an I2O OS specific module that supports</td>
</tr>
<tr>
<td><strong>icmp(7P)</strong></td>
<td>Internet Control Message Protocol</td>
</tr>
<tr>
<td><strong>iee(7D)</strong></td>
<td>Intel EtherExpress 16 Ethernet device driver</td>
</tr>
<tr>
<td><strong>ieef(7D)</strong></td>
<td>Intel EtherExpress Flash32/82596 Ethernet device driver</td>
</tr>
<tr>
<td><strong>if(7P)</strong></td>
<td>See <strong>if_tcp(7P)</strong></td>
</tr>
</tbody>
</table>

Last modified 29 Sep 1994

SunOS 5.7
**Device and Network Interfaces**

- `if_tcp(7P)` - general properties of Internet Protocol network interfaces
- `ifp(7D)` - ISP2100 Family Fibre Channel Host Bus Adapter Driver
- `inet(7P)` - Internet protocol family
- `intro(7)` - See `Intro(7)`
- `ip(7P)` - Internet Protocol
- `ipd(7M)` - See `ppp(7M)`
- `ipdc(7M)` - See `ppp(7M)`
- `ipdptp(7M)` - See `ppp(7M)`
- `iprb(7D)` - Intel 82557 (D100)-controlled Network Cards
- `isdnio(7I)` - ISDN interfaces
- `isp(7D)` - ISP SCSI Host Bus Adapter Driver
- `kb(7M)` - keyboard STREAMS module
- `kdmouse(7D)` - built-in mouse device interface
- `keyboard(7D)` - system console keyboard
- `kmem(7D)` - See `mem(7D)`
- `kstat(7D)` - kernel statistics driver
- `ksyms(7D)` - kernel symbols
- `ldterm(7M)` - standard STREAMS terminal line discipline module
- `le(7D)` - Am7990 (LANCE) Ethernet device driver
- `lebuffer(7D)` - See `le(7D)`
- `ledma(7D)` - See `le(7D)`
- `llc1(7D)` - Logical Link Control Protocol Class 1 Driver
- `lockstat(7D)` - kernel lock statistics driver
- `lofs(7FS)` - loopback virtual file system
<table>
<thead>
<tr>
<th>Device or Network Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log(7D)</td>
<td>interface to STREAMS error logging and event tracing</td>
</tr>
<tr>
<td>logi(7D)</td>
<td>LOGITECH Bus Mouse device interface</td>
</tr>
<tr>
<td>lp(7D)</td>
<td>driver for parallel port</td>
</tr>
<tr>
<td>ltem(7D)</td>
<td>ANSI Layered Console Driver</td>
</tr>
<tr>
<td>m64(7D)</td>
<td>8-bit PCI color memory frame buffer</td>
</tr>
<tr>
<td>mem(7D)</td>
<td>physical or virtual memory</td>
</tr>
<tr>
<td>mhd(7I)</td>
<td>multihost disk control operations</td>
</tr>
<tr>
<td>mic(7D)</td>
<td>Multi-interface Chip driver</td>
</tr>
<tr>
<td>mlx(7D)</td>
<td>low-level module for Mylex DAC960E EISA and Mylex DAC960P/PD/PD-Ultra/PL PClhost bus adapter series</td>
</tr>
<tr>
<td>msm(7D)</td>
<td>Microsoft Bus Mouse device interface</td>
</tr>
<tr>
<td>mt(7D)</td>
<td>tape interface</td>
</tr>
<tr>
<td>mtio(7I)</td>
<td>general magnetic tape interface</td>
</tr>
<tr>
<td>ncrs(7D)</td>
<td>low-level module for NCR 53C710, 53C810, 53C815, 53C820, and 53C825 host bus adapters</td>
</tr>
<tr>
<td>nes(7D)</td>
<td>Novell NE3200 Ethernet device Driver</td>
</tr>
<tr>
<td>nei(7D)</td>
<td>Novell NE2000, NE2000plus Ethernet device Driver</td>
</tr>
<tr>
<td>nfe(7D)</td>
<td>Compaq Netflex-2 Dualport Ethernet and ENET/TR Drivers</td>
</tr>
<tr>
<td>null(7D)</td>
<td>the null file</td>
</tr>
<tr>
<td>openprom(7D)</td>
<td>PROM monitor configuration interface</td>
</tr>
<tr>
<td>pcata(7D)</td>
<td>PCMCIA ATA card device driver</td>
</tr>
<tr>
<td>pcelx(7D)</td>
<td>3COM EtherLink III PCMCIA Ethernet Adapter</td>
</tr>
<tr>
<td>pcfs(7FS)</td>
<td>DOS formatted file system</td>
</tr>
<tr>
<td>pcic(7D)</td>
<td>Intel i82365SL PC Card Interface Controller</td>
</tr>
<tr>
<td>pckt(7M)</td>
<td>STREAMS Packet Mode module</td>
</tr>
<tr>
<td>Device Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>pcmem(7D)</td>
<td>PCMCIA memory card nexus driver</td>
</tr>
<tr>
<td>pcn(7D)</td>
<td>AMD PCnet Ethernet controller device driver</td>
</tr>
<tr>
<td>pcram(7D)</td>
<td>PCMCIA RAM memory card device driver</td>
</tr>
<tr>
<td>pcscsi(7D)</td>
<td>low-level module for the AMD PCscsi, PCscsi II, PCnet-SCSI, and Qlogic QLA510 PCI-to-SCSI bus adapters</td>
</tr>
<tr>
<td>pcser(7D)</td>
<td>PCMCIA serial card device driver</td>
</tr>
<tr>
<td>pe(7D)</td>
<td>Xircom Pocket Ethernet device driver</td>
</tr>
<tr>
<td>pfmod(7M)</td>
<td>STREAMS Packet Filter Module</td>
</tr>
<tr>
<td>pipemod(7M)</td>
<td>STREAMS pipe flushing module</td>
</tr>
<tr>
<td>pln(7D)</td>
<td>SPARCstorage Array SCSI Host Bus Adapter Driver</td>
</tr>
<tr>
<td>pm(7D)</td>
<td>power management driver</td>
</tr>
<tr>
<td>pmc(7D)</td>
<td>Platform Management Chip driver</td>
</tr>
<tr>
<td>ppp(7M)</td>
<td>STREAMS modules and drivers for the Point-to-Point Protocol</td>
</tr>
<tr>
<td>ppp_diag(7M)</td>
<td>See ppp(7M)</td>
</tr>
<tr>
<td>ptem(7M)</td>
<td>STREAMS Pseudo Terminal Emulation module</td>
</tr>
<tr>
<td>ptm(7D)</td>
<td>STREAMS pseudo-tty master driver</td>
</tr>
<tr>
<td>pts(7D)</td>
<td>STREAMS pseudo-tty slave driver</td>
</tr>
<tr>
<td>pty(7D)</td>
<td>pseudo-terminal driver</td>
</tr>
<tr>
<td>qe(7D)</td>
<td>QEC/MACE Ethernet device driver</td>
</tr>
<tr>
<td>qec(7D)</td>
<td>QEC bus nexus device driver</td>
</tr>
<tr>
<td>qfe(7d)</td>
<td>SUNW,qfe Quad Fast-Ethernet device driver</td>
</tr>
<tr>
<td>quotactl(7I)</td>
<td>manipulate disk quotas</td>
</tr>
<tr>
<td>rns_smt(7D)</td>
<td>Rockwell Station Management driver</td>
</tr>
<tr>
<td>route(7P)</td>
<td>kernel packet forwarding database</td>
</tr>
<tr>
<td>Device and Network Interfaces</td>
<td>Intro(7)</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>----------</td>
</tr>
<tr>
<td><strong>rout</strong>ing(7P)</td>
<td>system support for packet network routing</td>
</tr>
<tr>
<td><strong>sad</strong>(7D)</td>
<td>STREAMS Administrative Driver</td>
</tr>
<tr>
<td><strong>sbpro</strong>(7D)</td>
<td>Sound Blaster Pro, Sound Blaster 16, and Sound Blaster AWE32 audio device driver</td>
</tr>
<tr>
<td><strong>sd</strong>(7D)</td>
<td>driver for SCSI disk and (ATAPI/SCSI) CD-ROM devices</td>
</tr>
<tr>
<td><strong>se</strong>(7D)</td>
<td>Siemens 82532 ESCC serial communications driver</td>
</tr>
<tr>
<td><strong>se_hd</strong>lc(7D)</td>
<td>on-board high-performance serial HDLC interface</td>
</tr>
<tr>
<td><strong>ses</strong>(7D)</td>
<td>SCSI enclosure services device driver</td>
</tr>
<tr>
<td><strong>sesio</strong>(7I)</td>
<td>enclosure services device driver interface</td>
</tr>
<tr>
<td><strong>sf</strong>(7D)</td>
<td>SOC+ FC-AL FCP Driver</td>
</tr>
<tr>
<td><strong>smartii</strong>(7D)</td>
<td>Compaq Smart-2 EISA/PCI and Smart-2SL PCI Array Controller driver</td>
</tr>
<tr>
<td><strong>smc</strong>(7D)</td>
<td>SMC 8003/8013/8216/8416 Ethernet device driver</td>
</tr>
<tr>
<td><strong>smce</strong>(7D)</td>
<td>SMC 3032/EISA dual-channel Ethernet device driver</td>
</tr>
<tr>
<td><strong>smceu</strong>(7D)</td>
<td>SMC Elite32 Ultra (8232) Ethernet device driver</td>
</tr>
<tr>
<td><strong>smcf</strong>(7D)</td>
<td>SMC Ether100 (9232) Ethernet device driver</td>
</tr>
<tr>
<td><strong>soc</strong>(7D)</td>
<td>Serial Optical Controller (SOC) device driver</td>
</tr>
<tr>
<td><strong>socal</strong>(7D)</td>
<td>Serial Optical Controller for Fibre Channel Arbitrated Loop (SOC+) device driver</td>
</tr>
<tr>
<td><strong>sockio</strong>(7I)</td>
<td>ioctls that operate directly on sockets</td>
</tr>
<tr>
<td><strong>ssd</strong>(7D)</td>
<td>driver for SPARCstorage Array and Fibre Channel Arbitrated Loop disk devices</td>
</tr>
<tr>
<td><strong>st</strong>(7D)</td>
<td>driver for SCSI tape devices</td>
</tr>
<tr>
<td><strong>stc</strong>(7D)</td>
<td>Serial Parallel Communications driver for SBus</td>
</tr>
<tr>
<td><strong>stp4020</strong>(7D)</td>
<td>STP 4020 PCMCIA Adapter</td>
</tr>
<tr>
<td><strong>streamio</strong>(7I)</td>
<td>STREAMS ioctl commands</td>
</tr>
</tbody>
</table>
sxp(7D) Rockwell 2200 SNAP Streams Driver

tcp(7P) Internet Transmission Control Protocol

tcx(7D) 24-bit SBus color memory frame buffer

termio(7I) general terminal interface

termiox(7I) extended general terminal interface

ticlts(7D) loopback transport providers

ticots(7D) See ticlts(7D)

ticotsord(7D) See ticlts(7D)

timod(7M) Transport Interface cooperating STREAMS module

tirdwr(7M) Transport Interface read/write interface STREAMS module

tmpfs(7FS) memory based filesystem

tpf(7D) Platform Specific Module (PSM) for Tricord Systems

Enterprise Server Models ES3000, ES4000 and ES5000.

tr(7D) IBM 16/4 Token Ring Network Adapter device driver

trantor(7D) low-level module for Trantor T348 Parallel SCSI host bus

adapter

ttcompat(7M) V7, 4BSD and XENIX STREAMS compatibility module

tty(7D) controlling terminal interface

udp(7P) Internet User Datagram Protocol

uscsi(7I) user SCSI command interface

visual_io(7I) Solaris VISUAL I/O control operations

volfs(7FS) Volume Management file system

vuid2ps2(7M) See vuidmice(7M)

vuid3ps2(7M) See vuidmice(7M)

vuidm3p(7M) See vuidmice(7M)
**Device and Network Interfaces**

- **vuidm4p(7M)**: See **vuidmice(7M)**
- **vuidm5p(7M)**: See **vuidmice(7M)**
- **vuidmice(7M)**: converts mouse protocol to Firm Events
- **wscons(7D)**: workstation console
- **xd(7D)**: disk driver for Xylogics 7053 SMD Disk Controller
- **xdc(7D)**: See **xd(7D)**
- **xt(7D)**: driver for Xylogics 472 1/2 inch tape controller
- **xy(7D)**: disk driver for Xylogics 450 and 451 SMD Disk Controllers
- **xyc(7D)**: See **xy(7D)**
- **zero(7D)**: source of zeroes
- **zs(7D)**: Zilog 8530 SCC serial communications driver
- **zsh(7D)**: On-board serial HDLC/SDLC interface

---

Last modified 29 Sep 1994

SunOS 5.7

12
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 2940, 2940W, 2940U, 2940UW, 3940, and 3940W, as well as motherboards with embedded AIC-7870P and AIC-7880P SCSI chips.

The complete list of support devices (see NOTES) is:

<table>
<thead>
<tr>
<th>Device 1</th>
<th>Device 2</th>
<th>Device 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIC-7560</td>
<td>AIC-7870</td>
<td>AIC-7881</td>
</tr>
<tr>
<td>AIC-7850</td>
<td>AIC-7871</td>
<td>AIC-7882</td>
</tr>
<tr>
<td>AIC-7855</td>
<td>AIC-7872</td>
<td>AIC-7884</td>
</tr>
<tr>
<td>AIC-7860</td>
<td>AIC-7874</td>
<td>AIC-7885</td>
</tr>
<tr>
<td>AIC-7861</td>
<td>AIC-7875</td>
<td></td>
</tr>
<tr>
<td>AIC-7862</td>
<td>AIC-7880</td>
<td></td>
</tr>
</tbody>
</table>

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.

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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO

attributes(5)

Information Library for Solaris 2.6 (Intel Platform Edition)

NOTES

Throughout the release, support of additional devices may be added. See the in the Information Library for Solaris 2.6 (Intel Platform Edition) for additional information.
The \texttt{adp} driver supports LUN (Logical Unit Number) values of 0 through 15, this is beyond the standard SCSI-2 requirements which call for support of LUNs 0 through 7.
aha(7D) Devices

NAME
aha – low-level module for Adaptec 154x ISA host bus adapters

DESCRIPTION
The aha module provides low-level interface routines between the common disk/tape I/O subsystem and the Adaptec ISA bus master 154x SCSI (Small Computer System Interface) controllers. The aha 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 it.

Board Configuration and Auto Configuration
The driver attempts to initialize itself in accordance with the information found in the configuration file, /kernel/drv/aha.conf. A list of the relevant user configurable items follows.

```
dma speed "dmaspeed=0"
bus on time "buson=5"
bus off time "busoff=9"
```

The I/O port is the ISA bus I/O address used for communication with the adapter. The direct memory access (DMA) channel should be set to the manufacturer’s default of 5 for the primary adapter. The DMA speed, bus on time, and bus off times may be set for optimum performance with each ISA motherboard. Refer to the Adaptec AHA-1540/1542 User’s Manual for instructions. All jumpers on the board should be set (or verified) to conform with the configuration file.

The 154xC and the 154xCF should be set to default values. Specifically, disable BIOS support for drives with more than 1024 cylinders and more than two BIOS drives. Make sure that the DMA transfer speed does not exceed the capabilities of the motherboard: most can not be run faster than the default 5.7.

The default configurations described in the Adaptec AHA-1540/1542 User’s Manual should be used for standard configurations of the system. If more than one board is to be used in a single system, each must at least occupy a different set of address ranges and use a different DMA channel. Use of a different interrupt level for each board is required.

The default listing of the configuration file is as follows:

```
dmaspeed=0 buson=5 busoff=5
flow_control="dmult" queue="qsort" disk="scdk" tape="sctp";
```
After installation, 154x controllers may be jumpered for any of the I/O address, IRQ, and DMA channel combinations supported by the hardware, provided that the parameters do not conflict with other devices on the system.

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

**SEE ALSO**

attributes(5)
<table>
<thead>
<tr>
<th>NAME</th>
<th>aic – low-level module for Adaptec AIC-6360 based ISA host bus adapters</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS</td>
<td>aic@rg</td>
</tr>
</tbody>
</table>
| DESCRIPTION| The `aic` module provides low-level interface routines between the common disk/tape I/O subsystem and AIC-6360 based SCSI (Small Computer System Interface) bus controllers. It also provides support for the Adaptec AHA-1510A, AHA-1520A, and AHA-1522A SCSI controllers. It supports the AIC-6360 SCSI controller on the Sound Blaster 16 SCSI-2 board but does not support the audio functions — the `sbpro` driver provides these capabilities. Note that the AHA-1510A board and the SCSI port on the Sound Blaster 16 SCSI-2 board can only be used as a secondary controller — not a primary (boot) controller.

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

The driver attempts to initialize itself using the information found in the configuration file, `aic.conf`.

If multiple boards are configured in a single system, each board must occupy a different I/O base address in the system I/O address space. Each board must also be assigned a different IRQ level.

The AHA-1522 or AHA-1520A controller can be used as a primary boot controller only at I/O base address 0x340 (unless special BIOS from Adaptec is procured). Therefore, if a system installation is performed using a SCSI device (such as a CD-ROM drive) connected to one of these adapters, the adapter must be configured with I/O base address 0x340. (The adapter BIOS supports booting from this address only.)

Refer to the AHA-1520A/1522A AT-to-SCSI Host Adapters Installation Guide provided with the controller for instructions on installation of the adapter and the valid jumper settings. The default jumper configuration described in the AHA-1520A/1522A AT-to-SCSI Host Adapters Installation Guide is the recommended configuration for the controller.

<table>
<thead>
<tr>
<th>FILES</th>
<th><code>kernel/drv/aic.conf</code>  aic device driver configuration file</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRIBUTES</td>
<td>See attributes(5) for descriptions of the following attributes:</td>
</tr>
</tbody>
</table>

SunOS 5.7

Last modified 13 May 1997
<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

**SEE ALSO**

- `driver.conf(4)`, `attributes(5)`, `sysbus(4)`, `sbpro(7D)`

**NOTES**

- The `aic` driver does not support direct memory access (DMA).
- The `aic` driver does not support SCSI timeouts.
arp(7P)

NAME
arp, ARP – Address Resolution Protocol

SYNOPSIS
#include <sys/fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/if_arp.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
10Mb/s Ethernet addresses. It is used by all the 10Mb/s Ethernet datalink
providers (interface drivers) and it can be used by other datalink providers
that support broadcast (such as FDDI and Token Ring). ARP is not specific to
the Internet Protocol but this implementation supports only that network layer
protocol.

ARP caches IP-to-Ethernet 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, the new mapping is
cached and any pending message is transmitted. ARP will queue at most four
packets while waiting for a mapping request to be responded to; only the four
most recently transmitted packets are kept.

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 which do not use ARP, ioctl()
requests are provided to enter and delete entries in the IP-to-Ethernet tables.

#include <sys/sockio.h>
#include <sys/socket.h>
#include <netinet/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);
Each ioctl() request takes the same structure as an argument. SIOCSARP sets an ARP entry, SIOCGARP gets an ARP entry, and SIOCDARP deletes an ARP entry. These ioctl() requests may be applied to any Internet family socket descriptor, or to a descriptor for the ARP device, but only by the privileged user.

The arpreq structure contains:

```c
/*
 * ARP ioctl request
 */
struct arpreq {
   struct sockaddr*arp_pa; /* protocol address */
   struct sockaddr*arp_ha; /* hardware address */
   int arp_flags; /* flags */
};
/* arp_flags field values */
#define ATF_COM 0x2 /* completed entry (arp_ha valid) */
#define ATF_PERM 0x4 /* permanent entry */
#define ATF_PUBL 0x8 /* publish (respond for other host) */
#define ATF_USETRAILERS 0x10 /* send trailer packets to host */
```

The address family for the arp_pa sockaddr must be AF_INET; for the arp_ha sockaddr it must be AF_UNSPEC. The only flag bits that may be written are ATF_PUBL and ATF_USETRAILERS. ATF_PERM makes the entry permanent if the ioctl() request succeeds. The peculiar nature of the ARP tables may cause the ioctl() request to fail if too many permanent IP addresses hash to the same slot. ATF_PUBL specifies that the ARP code should respond to ARP requests for the indicated host 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.

ARP is also used to negotiate the use of trailer IP encapsulations; trailers are an alternate encapsulation used to allow efficient packet alignment for large packets despite variable-sized headers. Hosts that wish to receive trailer encapsulations so indicate by sending gratuitous ARP translation replies along with replies to IP requests; they are also sent in reply to IP translation replies. The negotiation is thus fully symmetrical, in that either or both hosts may request trailers. The ATF_USETRAILERS flag is used to record the receipt of such a reply, and enables the transmission of trailer packets to that host.

ARP watches passively for hosts impersonating the local host (that is, a host which responds to an ARP mapping request for the local host’s address).

SEE ALSO

arp(1M), ifconfig(1M), if_tcp(7P), inet(7P)


**DIAGNOSTICS**

**IP:** Hardware address '%x:%x:%x:%x:%x:%x' trying to be our address '%d.%d.%d'

*Duplicate IP address. ARP has discovered another host on the local network which responds to mapping requests for the Internet address of this system.*

**IP:** Proxy ARP problem? Hardware address '%x:%x:%x:%x:%x:%x' thinks it is '

*This message will appear if 'arp (1M)' has been used to create a published 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/ttynn", mode);
open("/dev/ttydn", mode);
open("/dev/cuan", mode);

DESCRIPTION
The asy module is a loadable STREAMS driver that provides basic support for
the standard UARTS that use Intel-8250, National Semiconductor-16450/16550
hardware, together with basic asynchronous communication support. The
driver 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/tty00 and /dev/tty01 are used to
access the two standard serial ports (COM1 and COM2) on an x86 based
system. The asy driver supports up to four serial ports, including the standard
ports. These ttynn devices have minor device numbers in the range 00-03.

By convention these same devices may be given names of the form
/dev/ttydn, where n denotes which line is to be accessed. Such device names
are typically used to provide a logical access point for a dial-in line being used
with a modem.

To allow a singletty line to be connected to a modem and used for both
incoming and outgoing calls, a special feature, controlled by the minor device
number, is available. By accessing character-special devices with names of the
form /dev/cuann 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/cuann line is opened, the corresponding tty, or ttyd line cannot
be opened until the /dev/cuann line is closed; a blocking open will wait until
the /dev/cuann line is closed (which will drop Data Terminal Ready, after
which Carrier Detect will usually drop as well) and carrier is detected again,
and a non-blocking open will return an error. Also, if the /dev/ttydn line has
been opened successfully (usually only when carrier is recognized on the
modem) the corresponding /dev/cuann line cannot be opened. This allows a

Last modified 13 Feb 1997   SunOS 5.7   22
modem to be attached to, for example, /dev/ttyd0 (renamed from /dev/tty00) 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/cua0 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 of the speeds supported by termio. The speeds cannot be set independently; when the output speed is set, the input speed is set to the same speed.

ERRORS

An open() will fail if:

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 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/tty[00-03] hardwired tty lines
/dev/ttyd[0-3] dial-in tty lines
/dev/cua[0-3] dial-out tty lines
/platform/i86pc/kernel/drv/asy.conf asy configuration file

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO

tip(1), uucp(1C), ioctl(2), open(2), termios(3), attributes(5), ldterm(7M), termio(7I), ttcompat(7M)

23 SunOS 5.7 Last modified 13 Feb 1997
DIAGNOSTICS

asy\text{n}: silo overflow.  The hardware overrun occurred before the input character could be serviced.

asy\text{n}: ring buffer overflow.  The driver’s character input ring buffer overflowed before it could be serviced.
NAME
ata – AT attachment disk driver

SYNOPSIS
ata@1,ioaddr

DESCRIPTION
The ata driver supports disk and CD-ROM interfaces conforming to the AT Attachment specification including IDE interfaces. It excludes the MFM, RLL, ST506, and ST412 interfaces. Support is provided for CD_ROM drives that conform to the Small Form Factor (SFF) ATA Packet Interface (ATAPI) specification: SFF-8020 revision 1.2.

CONFIGURATION
The driver initializes itself in accordance with the information found in the configuration file ata.conf (see below). The only user configurable items in this file are:

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 the tuning be carried out using a duplicate of the /platform/i86pc/kernel directory subtree. This will ensure that a bootable kernel subtree exists in the event of a failed test.

max_transfer

This value controls the size of individual requests for consecutive disk sectors. The value may range from 0x1 to 0x100. Higher values yield higher throughput. The system is shipped with a value of 0x100, which probably should not be changed.
EXAMPLE 1  A sample display of ata configuration file.

The following is an example of an ata.conf configuration file.

```bash
# 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" ;
```

x86 FILES

/platform/i86pc/kernel/drv/ata The device file.
/platform/i86pc/kernel/drv/ata.conf The configuration file.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO

attributes(5), aha(7D), cmdk(7D), dpt(7D), eha(7D)
audio(7I)  Iocll Requests

### NAME
audio – generic audio device interface

### OVERVIEW
The audio interface described below is an uncommitted interface and may be replaced in the future.

An audio device is used to play and/or record a stream of audio data. Since a specific audio device may not support all of the 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.

### 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. In order 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 mu-law format (as in the Greek letter mu) 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 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 recording or playing.

<table>
<thead>
<tr>
<th><strong>Sample Rate</strong></th>
<th>Sample rate is a number that represents the sampling frequency (in samples per second) of the audio data.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Encodings</strong></td>
<td>An encoding parameter specifies the audio data representation. mu-law encoding (pronounced mew-law) 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 G.711, and is the standard encoding for telephony elsewhere in the world. A-law and mu-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 audio format in which sample values are directly proportional to audio signal voltages. Each</td>
</tr>
</tbody>
</table>

27 SunOS 5.7 Last modified 14 Jan 1998
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, mu-law and A-law data are stored with 8-bit precision. PCM data may be stored at various precisions, though 16-bit PCM is 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 2 16-bit samples, corresponding to the left and right channel data.

**DESCRIPTION**

The device `/dev/audio` is a device driver that dispatches audio requests to the appropriate underlying audio device driver. 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.

As 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 treated as an exclusive resource – only one process can open the device at a time. However, two processes may simultaneously access the device: if one opens it read-only, then another may open it write-only.

When a process cannot open `/dev/audio` because the requested access mode is busy:

- if either the `O_NDELAY` or `O_NONBLOCK` flag 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`. This allows a process to block in the `open` call, while waiting for the audio device to become available.

Upon the initial `open()` of the audio device, the driver will reset the data format of the device to the default state of 8-bit, 8Khz, mono mu-law data. If the device is already open and a different audio format has been set, this will not be possible. Audio applications should explicitly set the encoding characteristics to match the audio data requirements, rather than depend on the default configuration.
Since the audio device grants exclusive read or write access to a single process at a time, long-lived audio applications may choose to close the device when they enter an idle state and reopen it when required. The `play_waiting` and `record_waiting` flags in the audio information structure (see below) provide an indication that another process has requested access to the device. For instance, a background audio output process may choose to relinquish the audio device whenever another process requests write access.

### Recording Audio Data

The `read()` system call copies data from the system 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, mu-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 applications 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

---

SunOS 5.7

---

29

Last modified 14 Jan 1998
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 (see below) is incremented.

The final close(2) of the file descriptor hangs until 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 conversion 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.

Output data will be played from the STREAMS buffers at a rate of at least 8000 bytes per second for mu-law or A-law 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 will be generated and the device will need to be closed before any future writes will succeed.

Asynchronous I/O

The I_SETSIG STREAMS ioctl enables asynchronous notification, through the SIGPOLL signal, of input and output ready conditions. 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.
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 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/audioio.h>, as follows:

```c
/* 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 flags */
    uchar_t active; /* non-zero if I/O active */
    uint_t avail_ports; /* available I/O ports */
} audio_prinfo_t;
/* This structure is used in AUDIO_GETINFO and AUDIO_SETINFO ioctl commands */
typedef struct audio_info {
    audio_prinfo_t record; /* input status information */
    audio_prinfo_t play; /* output status information */
    uint_t monitor_gain; /* input to output mix */
    uchar_t output_muted; /* non-zero if output muted */
} audio_info_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. Most audio devices allow input data to be monitored by mixing audio input onto the output channel. The `monitor.gain` field controls the level of this feedback path.

The `play.port` field controls the output path for the audio device. It can be set to either `AUDIO_SPEAKER` (built-in speaker), `AUDIO_HEADPHONE` (headphone jack), or `AUDIO_LINE_OUT` (line-out port). For some devices, it may be set to a combination of these ports. The `play.avail_ports` field returns the set of output ports that are currently accessible. The input ports can be either `AUDIO_MICROPHONE` or `AUDIO_LINE_IN`. The `record.avail_ports` field returns the set of input ports that are currently accessible.

The `play.balance` and `record.balance` fields are used to control the volume between the left and right channels when manipulating stereo data. When the value is set between `AUDIO_LEFT_BALANCE` and `AUDIO_MID_BALANCE`, the right channel volume will be reduced in proportion to the `balance` value.
Conversely, when balance is set between AUDIO_MID_BALANCE and AUDIO_RIGHT_BALANCE, the left channel will be proportionally reduced.

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 initialized, at open(), to zero and increment 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 the 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 above. Refer to the audio device-specific manual pages for a list of supported data format combinations.

The data format fields may be modified only on the `/dev/audio` device. The audio hardware will often constrain the input and output data formats to be identical. If this is the case, then the data format may not be changed if multiple processes have opened the audio device.

If the parameter changes requested by an `AUDIO_SETINFO` ioctl cannot all be accommodated, `ioctl()` will return with `errno` set to EINVAL and no changes will be 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, or until a signal is delivered to the calling process. It may not be issued for the `/dev/audioctl` device. An implicit `AUDIO_DRAIN` is performed on the final `close()` of `/dev/audio`.

- **AUDIO_GETDEV**
  The argument is a pointer to an `audio_device` 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` 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.
The `AUDIO_SETINFO` ioctl requests a pointer to an `audio_info` 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 structure supplied and overwrites the 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` was issued. This allows programs to automatically modify these fields while retrieving the previous value.

Certain fields in the 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.

Once set, the following values persist through subsequent `open()` and `close()` calls of the device: `play.gain`, `record.gain`, `play.balance`, `record.balance`, `output_muted`, `monitor_gain`, `play.port`, and `record.port`. However, an automatic device driver unload will reset these parameters to their default values on the next load. All other state is reset when the corresponding I/O stream of `/dev/audio` is closed.

The `audio_info` 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:

```c
audio_info_t info;
AUDIO_INITINFO(&info);
info.play.port = AUDIO_HEADPHONE;
err = ioctl(audio_fd, AUDIO_SETINFO, &info);
```

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

SunOS 5.7	Last modified 14 Jan 1998
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.

EBUSY The parameter changes requested in the AUDIO_SETINFO ioctl could not be made because another process has the device open and is using a different format.

FILES

The physical audio device names are system dependent and are rarely used by programmers. The programmer should use the generic device names listed below.

/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

SEE ALSO

close(2), fcntl(2), ioctl(2), open(2), poll(2), read(2), write(2), audioamd(7D), audiocs(7D), dbri(7D), sbpro(7D), 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.

On LX machines running Solaris 2.3, catting a demo audio file to the audio device /dev/audio does not work. Use the audioplay command on LX machines instead of cat.

FUTURE DIRECTIONS

Future workstation audio resources will be managed by an audio foundation library. For the time being, we encourage you to write your programs in a modular fashion, isolating the audio device-specific functions, so that they may be easily ported to such an environment.

Last modified 14 Jan 1998
The AUDIO_GETDEV ioctl is provided for the future implementation of an audio device capability database. In general, applications may use the `play.avail_ports` and `record.avail_ports` fields of the `audio_info` structure to determine the audio device capabilities.
NAME

audioamd – telephone quality audio device

DESCRIPTION

The audioamd device uses the AM79C30A Digital Subscriber Controller chip to implement the audio device interface. This interface is described fully in the audio(7I) manual page.

Applications that open /dev/audio may use the AUDIO_GETDEV ioctl to determine which audio device is being used. The audioamd driver will return "SUNW,am79c30" in the name field of the audio_device structure. The version field will contain "a" and the config field will be set to "onboard1".

The AUDIO_SETINFO ioctl controls device configuration parameters. When an application modifies the record.buffer_size field using the AUDIO_SETINFO ioctl, the driver will constrain it to be greater than zero and less than or equal to 8000 bytes or one second of audio data. Applications are warned that setting this field too low or too high may cause system performance problems and should therefore set this field with caution.

Audio Data Formats

The audioamd device supports the audio formats listed in the following table. When the device is open for simultaneous play and record, the input and output data formats must match.

<table>
<thead>
<tr>
<th>Sample Rate</th>
<th>Encoding</th>
<th>Precision</th>
<th>Channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>8000 Hz</td>
<td>mu-law (as in the Greek letter mu)</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>8000 Hz</td>
<td>A-law</td>
<td>8</td>
<td>1</td>
</tr>
</tbody>
</table>

Since audioamd supports only single-channel (monaural) audio, the play.balance and record.balance fields of the audio_info structure are ignored.

Audio Ports

The recordavail_ports and playavail_ports fields of the audio_info structure report the available input and output ports. The audioamd device supports one input port, selected by setting the record.port field to AUDIO_MICROPHONE. The play.port field may be set to either AUDIO_SPEAKER or AUDIO_HEADPHONE, to direct audio output to the built-in speaker or headphone jack, respectively. Note that AUDIO_SPEAKER cannot be enabled for systems that do not include a built-in speaker.

Sample Granularity

Since the audioamd device manipulates single samples of audio data, the reported input and output sample counts will be very close to the actual sample count. However, some other audio devices report sample counts that are approximate, due to buffering constraints. Programs should, in general, not rely on absolute accuracy of the sample count fields.
FILES
/dev/audio
/dev/audiocntl
/dev/sound
/usr/demo/SOUND

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>SPARC: SPARCstation 1 and 2,</td>
</tr>
<tr>
<td></td>
<td>IPC, IPX, SLC, ELC, LC,</td>
</tr>
<tr>
<td></td>
<td>and SPARCserver 6xx systems</td>
</tr>
</tbody>
</table>

Desktop SPARC systems include a built-in speaker for audio output. The audio cable provides connectors for a microphone and external headset. The headset output level is adequate to power most headphones, but may be too low for some external speakers. Powered speakers or an external amplifier may be used. SPARCserver 6xx systems do not have an internal speaker, but support an external microphone and speaker connected through the audio cable.

The Sun Microphone is recommended for normal desktop audio recording. It contains a battery that must be replaced after 210 hours of use. Other microphones may be used, but a pre-amplifier circuit may be required to achieve a sufficient input signal. Other audio sources may be recorded by connecting one channel of the line output to the audio cable microphone input. If the input signal is distorted, external attenuation may be required (audio sources may also be connected from their headphone output with the volume turned down).

SEE ALSO ioctl(2), attributes(5), audio(7I), streamio(7I)

AMD data sheet for the AM79C30A Digital Subscriber Controller, Publication number 09893.
NAME      audiocs – Crystal Semiconductor 4231 audio Interface

DESCRIPTION     Audiocs is an audio interface that provides line in and line out ports for audio devices. It can be either an integrated device or an add-in option card.

SPARC     SPARCstation 5 systems have the Multimedia Codec integrated onto the CPU board of the machine. In the "onboard" Codec, there are microphone, line in, headphone, and line out ports located on the system back panel. In addition, the headphone and microphone ports do not have the input detection circuitry to determine whether or not there is currently headphones or a microphone plugged in. There is no interface on the SPARCstation 5 for the speakerbox to connect to.

     SPARCstation 4 systems have ports for microphone, line in, headphone, and line out, as well as a port for an internal CD-ROM.

     For all SPARCstations, the new Sun Microphone II is recommended for normal desktop audio recording. Other audio sources may be recorded by connecting their line output to the line input (audio sources may also be connected from their headphone output if the volume is adjusted properly).

Ultra     Ultra systems have ports for microphone, line in, headphone and line out.

x86      The Multimedia Codec may be found as either an integrated motherboard device, or as an add-in option card. An internal CD-ROM is also a common input option.

APPLICATION PROGRAM INTERFACE

Applications that open /dev/audio may use the AUDIO_GETDEV ioctl to determine which audio device is being used. The audiocs driver will return the string "SUNW,CS4231" in the name field of the audio_device structure. The version field will contain one of the following values, depending upon the platform:

<table>
<thead>
<tr>
<th>Platform Type</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPARCstation 4/5</td>
<td>a</td>
</tr>
<tr>
<td>Ultra</td>
<td>b</td>
</tr>
<tr>
<td>reserved</td>
<td>c</td>
</tr>
</tbody>
</table>

The config field will contain the following value: "onboard1" on a /dev/audio stream associated with the onboard Multimedia Codec.

The AUDIO_SETINFO ioctl controls device configuration parameters. When an application modifies the record.buffer_size field using the AUDIO_SETINFO ioctl, the driver will constrain it to be non-zero and up to a maximum of 8180 bytes.
Audio Data Formats

The Multimedia 4231 Codec audiocs device supports the audio formats listed in the following table. When the device is open for simultaneous play and record, the input and output data formats must match.

<table>
<thead>
<tr>
<th>Sample Rate</th>
<th>Encoding</th>
<th>Precision</th>
<th>Channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>8000 Hz</td>
<td>mu-law (as in the Greek letter mu) or A-law</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>9600 Hz</td>
<td>mu-law or A-law</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>11025 Hz</td>
<td>mu-law or A-law</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>16000 Hz</td>
<td>mu-law or A-law</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>18900 Hz</td>
<td>mu-law or A-law</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>22050 Hz</td>
<td>mu-law or A-law</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>32000 Hz</td>
<td>mu-law or A-law</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>37800 Hz</td>
<td>mu-law or A-law</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>44100 Hz</td>
<td>mu-law or A-law</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>48000 Hz</td>
<td>mu-law or A-law</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>8000 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
</tr>
<tr>
<td>9600 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
</tr>
<tr>
<td>11025 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
</tr>
<tr>
<td>16000 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
</tr>
<tr>
<td>18900 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
</tr>
<tr>
<td>22050 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
</tr>
<tr>
<td>32000 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
</tr>
<tr>
<td>37800 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
</tr>
<tr>
<td>44100 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
</tr>
<tr>
<td>48000 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
</tr>
</tbody>
</table>

Audio Ports

The record.avail_ports and play.avail_ports fields of the audio_info structure report the available input and output ports. In most environments, the audiocs device supports three input ports, except the Ultra product family which supports only two. These input ports are selected by setting the record.port field to either AUDIO_MICROPHONE, AUDIO_LINE_IN, or AUDIO_INTERNAL_CD_IN. (Ultra
systems do not support AUDIO_INTERNAL_CD_IN.) If you select the
AUDIO_INTERNAL_CD_IN this will select input from the internal CD drive, if
present. This will allow you to gather data from the CD without having to
hook up a connecting line from the headphone jack to the line input jack. The
play.port field may be set to any combination of AUDIO_SPEAKER,
AUDIO_HEADPHONE, and AUDIO_LINE_OUT by OR’ing the desired port names
together. (Note: On some systems, the headphone and line out ports internally
share the same circuitry; in these cases, it is not possible to enable either
output exclusively.)

Sample Granularity
Since the audiocs device manipulates buffers of audio data, at any given time
the reported input and output sample counts will vary from the actual sample
count by no more than the size of the buffers it is transferring. Programs
should, in general, not rely on absolute accuracy of the play.samples and
record.samples fields of the audio_info structure.

Audio Status Change
Notification
As described in audio(7I), it is possible to request asynchronous notification
of changes in the state of an audio device.

ERRORS
audiocs errors are defined in the audio(7I), man pages.

FILES
The physical device names are very system dependent and are rarely used by
programmers.

SPARC Only
/devices/iommu@f,e0000000/sbus@f,e0001000/SUNW,CS4231@2,c00000:sound,audio

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>SPARCstation 4/5, x86</td>
</tr>
</tbody>
</table>

SEE ALSO
ioct1(2), attributes(5), audio(7I), streamio(7I)

Crystal Semiconductor, Inc., data sheet for the CS4231 16-Bit, 48 kHz,
Multimedia Audio Codec Publication number DS111PP2.

NOTES.
The AUDIO_INTERNAL_CD_IN is another new functionality addition. Because
of this, audiotool will now have a new button appear in the record popup
box that will allow the user of audiotool to switch to the internal CD (if
present).
NAME
bd – SunButtons and SunDials STREAMS module

SYNOPSIS
open("/dev/bd", O_RDWR)

DESCRIPTION
The bd STREAMS module processes the byte streams generated by the
SunButtons buttonbox and SunDials dialbox. The buttonbox generates a
stream of bytes that encode the identity and state transition of the buttons. The
dialbox generates a stream of bytes that encode the identity of the dials and the
amount by which they are turned. Both of these streams are merged together
when a host has both a buttonbox and a dialbox in use at the same time.

SunButtons reports the button number and up/down status encoded into a
one byte message. Byte values from 0xc0 to 0xdf indicate a transition to button
down. To obtain the button number, subtract 0xc0 from the byte value. Byte
values from 0xe0 to 0xff indicate a transition to button up. To obtain the button
number, subtract 0xe0 from the byte value.

Each dial sample in the byte stream consists of three bytes. The first byte
identifies which dial was turned and the next two bytes return the delta in
signed binary format. When bound to an application using the window
system, Virtual User Input Device events are generated. An event from a dial
is constrained to lie between 0x80 and 0x87.

A stream with the bd pushed streams module configured in it can emit
firm_events as specified by the protocol of a VUID. bd understands the
VUIDSFORMAT and VUIDGFORMAT ioctl (see reference below), as defined in
/usr/include/sys/bdio.h and
$OPENWINHOME/include/xview/win_event.h. All other ioctl() requests
are passed downstream.

The bd streams module sets the parameters of the serial port when it is first
opened. No termio(7I) ioctl () requests should be performed on a bd
STREAMS module, as bd expects the device parameters to remain as it set
them.

IOCTLS
VUIDSFORMAT
VUIDGFORMAT
These are standard Virtual User Input Device ioctl.

BDIOBUTLITE
The bd streams module implements this ioctl to
enable processes to manipulate the lights on the
buttonbox. The BDIOBUTLITE ioctl must be
carried by an I_STR ioctl to the bd module. For
an explanation of I_STR see streamio(7I). The
data for the BDIOBUTLITE ioctl is an unsigned
integer in which each bit represents the lamp on one button. The macro LED_MAP in <sys/bdio.h> maps button numbers to appropriate bits. Source code for the demo program x_buttontest is provided with the buttons and dials package, and may be found in the directory /usr/demo/BUTTONBOX. Look at x_buttontest.c for an example of how to manipulate the lights on the buttonbox.

FILES
/usr/include/sys/bdio.h
/usr/include/sys/stropts.h
$OPENWINHOME/share/include/xview/win_event.h

SEE ALSO
bdconfig(1M), ioctl(2), x_buttontest(6), x_dialtest(6), streamio(7I), termio(7I)

SunButtons Installation and Programmers Guide SunDials Installation and Programmers Guide

WARNINGS
The SunDials dial box must be used with a serial port.
NAME
bpp – bi-directional parallel port driver

SYNOPSIS
SUNW, bpp@slot, offset : bppn

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

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 occurs.

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
Devices

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.
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.

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

```c
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 */
    BPP_CLEAR_MEM, /* write 0’s to memory */
    BPP_SET_MEM, /* write 1’s to memory */
};
```

47 SunOS 5.7 Last modified 22 Aug 1994
/* The following handshakes are RESERVED. Do not use. */
#define BPP_VPRINT_HS /* valid only in read/write mode */
#define BPP_VPLOT_HS /* valid only in read/write mode */
};

The read_setup_time field controls the time between dsr/f 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 dsr/f 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 dsr/f 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 dsr/f rising edge to dsr/f
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 dsr/f
rising edge for all values of the write_handshake field.

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

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_SLCTPIN_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_PIN 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 */

Transfer Pins
Structure

Last modified 22 Aug 1994
**Error Pins Structure**

This structure is defined in the include file `<sys/bpp_io.h>`.

```c
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** `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). In read-only mode, with the read_handshake set to VPRINT or VPLOT, an INIT signal has been received, and another transfer has been attempted.

**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.
Devices

FILES
/dev/bppn  bi-directional parallel port devices

SEE ALSO  ioctl(2), read(2), write(2), sbus(4)
bufmod(7M) STREAMS Modules

NAME
bufmod – STREAMS Buffer Module

SYNOPSIS
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
bufmod’s behavior 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. bufmod’s default settings 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 SBIOC(snapshot length) 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 timeval132 sbh_timestamp;
    #endif
```
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 \texttt{SB\_DEFER\_CHUNK} flag is set, buffering does not begin until the second message is received within the timeout window.

If the \texttt{SB\_SEND\_ON\_WRITE} flag is set, \texttt{bufmod} passes up the read side any buffered data when a message is received on the write side. \texttt{SB\_SEND\_ON\_WRITE} and \texttt{SB\_DEFER\_CHUNK} are often used together. 

Write-side Behavior 

\texttt{bufmod} intercepts \texttt{M\_IOCTL} messages for the \texttt{ioctl}s described below. The module passes all other messages through unaltered to its lower neighbor. If \texttt{SB\_SEND\_ON\_WRITE} is set, message arrival on the writer side suffices to close and transmit the current read side chunk.

IOCTLS

\texttt{bufmod} responds to the following \texttt{ioctl}s.

\texttt{SBIOCSTIME} Set the read timeout value to the value referred to by the \texttt{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 \texttt{EINVAL} error.

\texttt{SBIOCGTIME} Return the read timeout in the \texttt{struct timeval} pointed to by the argument. If the timeout has been cleared with the \texttt{SBIOCCTIME} \texttt{ioctl}, return with an \texttt{ERANGE} error.

\texttt{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.

\texttt{SBIOCSCHUNK} Set the chunk size to the value referred to by the \texttt{uint\_t} pointer given as argument. See \texttt{NOTES} for description of effect on stream head high water mark.

\texttt{SBIOCGCHUNK} Return the chunk size in the \texttt{uint\_t} pointed to by the argument.

\texttt{SBIOCSSNAP} Set the current snapshot length to the value given in the \texttt{uint\_t} pointed to by the \texttt{ioctl}'s final argument. \texttt{bufmod} interprets a snapshot length value of zero as meaning infinity, so it will not
<table>
<thead>
<tr>
<th>Stream Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBIOCGSNAP</td>
<td>Returns the current snapshot length in the <code>uint_t</code> pointed to by the <code>ioctl</code>'s final argument.</td>
</tr>
<tr>
<td>SBIOCSFLAGS</td>
<td>Set the current flags to the value given in the <code>uint_t</code> pointed to by the <code>ioctl</code>'s final argument. Possible values are a combination of the following:</td>
</tr>
<tr>
<td></td>
<td>SB_SEND_ON_WRITE: Transmit the read side chunk on arrival of a message on the write side.</td>
</tr>
<tr>
<td></td>
<td>SB_NO_HEADER: Do not add headers to read side messages.</td>
</tr>
<tr>
<td></td>
<td>SB_NO_DROPS: Do not drop messages due to flow control upstream.</td>
</tr>
<tr>
<td></td>
<td>SB_NO_PROTO_CVT: Do not convert <code>M_PROTO</code> messages into <code>M_DATA</code>.</td>
</tr>
<tr>
<td></td>
<td>SB_DEFER_CHUNK: Begin buffering on arrival of the read side message in a timeout interval.</td>
</tr>
</tbody>
</table>

**SEE ALSO**

dlpi(7P), ie(7D), le(7D), 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
bwtwo – black and white memory frame buffer

## SYNOPSIS
```
/dev/fbs/bwtwo
```

## DESCRIPTION
The `bwtwo` interface provides access to monochrome memory frame buffers. It supports the ioctls described in `fbio(7I)`.

Reading or writing to the frame buffer is not allowed — you must use the `mmap(2)` system call to map the board into your address space.

## FILES
```
/dev/fbs/bwtwo[0-9]  device files
```

## SEE ALSO
`mmap(2), cgfour(7D), fbio(7I)`

## BUGS
Use of vertical-retrace interrupts is not supported.
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.

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
 */
ioctl Requests

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

Last modified 18 Feb 1998
ioctl Requests

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*/
};
```
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.

```c
/*
 * 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

Last modified 18 Feb 1998

SunOS 5.7
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.

```c
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.
Ioctl Requests

<table>
<thead>
<tr>
<th>CDROM_AUDIO_COMPLETED</th>
<th>Audio play successfully completed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDROM_AUDIO_ERROR</td>
<td>Audio play stopped due to error.</td>
</tr>
<tr>
<td>CDROM_AUDIO_NO_STATUS</td>
<td>No current audio status to return.</td>
</tr>
</tbody>
</table>

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, 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;
};
```

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. |

Last modified 18 Feb 1998
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, 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()`.

```c
/*
 * 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:
**ioctl Requests**

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

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 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.
CDROMBLKMODE

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.
CDROM_DRV_SPEED

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").


SCSI-2 Standard, document X3T9.2/86-109
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.

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
cgeight – 24-bit color memory frame buffer

SYNOPSIS
/dev/fbs/cgeight

DESCRIPTION
The cgeight is a 24-bit color memory frame buffer with a monochrome overlay plane and an overlay enable plane implemented optionally on the Sun-4/110, Sun-4/150, Sun-4/260 and Sun-4/280 system models. It provides the standard frame buffer interface as defined in fbio(7I).

In addition to the ioctls described under fbio(7I) the cgeight interface responds to two cgeight-specific colormap ioctls, FBIOPUTCMP and FBIOGETCMAP. FBIOPUTCMP returns no information other than success/failure using the ioctl return value. FBIOGETCMAP returns its information in the arrays pointed to by the red, green, and blue members of its fbcmap structure argument; fbcmap is defined in <sys/fbio.h> as:

```c
struct fbcmap {
    int index;       /* first element (0 origin) */
    int count;       /* number of elements */
    unsigned char *red;        /* red color map elements */
    unsigned char *green;      /* green color map elements */
    unsigned char *blue;       /* blue color map elements */
};
```

The driver uses color board vertical-retrace interrupts to load the colormap.

The systems have an overlay plane colormap, which is accessed by encoding the plane group into the index value with the PIX_GROUP macro (see <sys/pr_planegroups.h>).

When using the mmap(2) system call to map in the cgeight frame buffer. The device looks like:

```
DACBASE: 0x200000 -> Brooktree Ramdac 16 bytes
         0x202000 -> P4 Register 4 bytes
OVLBASE: 0x210000 -> Overlay Plane 1152x900x1
         0x230000 -> Overlay Enable Planea 1152x900x1
         0x250000 -> 24-bit Frame Buffera 1152x900x32
```

FILES
/dev/fbs/cgeight
<sys/fbio.h>
<sys/pr_planegroups.h>
SEE ALSO  mmap(2), fbio(7I)
NAME
cgfour - P4-bus 8-bit color memory frame buffer

SYNOPSIS
/dev/fbs/cgfour

DESCRIPTION
The cgfour is a color memory frame buffer with a monochrome overlay plane and an overlay enable plane. It provides the standard frame buffer interface as defined in fbio(7I).

In addition to the ioctls described under fbio(7I) the cgfour interface responds to two cgfour-specific colormap ioctls, FBIOPUTCMAP and FBIOGETCMAP. FBIOPUTCMAP returns no information other than success/failure using the ioctl return value. FBIOGETCMAP returns its information in the arrays pointed to by the red, green, and blue members of its fbcmap structure argument; fbcmap is defined in <sys/fbio.h> as:

```c
struct fbcmap {
    int index; /* first element (0 origin) */
    int count; /* number of elements */
    unsigned char *red; /* red color map elements */
    unsigned char *green; /* green color map elements */
    unsigned char *blue; /* blue color map elements */
};
```

The driver uses color board vertical-retrace interrupts to load the colormap.

The cgfour has an overlay plane colormap, which is accessed by encoding the plane group into the index value with the PIX_GROUP macro (see <sys/pr_planegroups.h>).

FILES
/dev/fbs/cgfour0

SEE ALSO
mmap(2), fbio(7I)
NAME
cfourseen – 24-bit color graphics device

SYNOPSIS
/dev/fbs/cfourseen

DESCRIPTION
The cfourseen device driver controls the video SIMM (VSIMM) component of the video and graphics subsystem of the Desktop SPARCsystems with SX graphics option. The VSIMM provides 24-bit truecolor visuals in a variety of screen resolutions and pixel depths.

The driver supports multi-threaded applications and has an interface accessible through mmap(2). The user must have an effective user ID of 0 to be able to write to the control space of the cfourseen device.

There are eight distinct physical spaces the user may map, in addition to the control space. The mappings are set up by giving the desired offset to the mmap(2) call.

The cfourseen device supports the standard frame buffer interface as defined in fbio(7I).

The cfourseen device can serve as a system console device.

See /usr/include/sys/cf14io.h for other device-specific information.

FILES
/kernel/drvcfourseen cfourseen device driver
/dev/fbs/cfourseen[0-9] Logical device name.
/usr/include/sys/cf14io.h Header file that contains device specific information
/usr/include/sys/cf14reg.h Header file that contains device specific information

SEE ALSO
mmap(2), fbio(7I)
NAME
cgsix – accelerated 8-bit color frame buffer

SYNOPSIS
/dev/fbs/cgsixn

DESCRIPTION
cgsix is a low-end graphics accelerator designed to enhance vector and
gon drawing performance. It has an 8-bit color frame buffer and provides
the standard frame buffer interface as defined in fbio(7I).

In addition, cgsix supports the following cgsix-specific IOCTL, defined in
/sys/fbio.h.

FBIOGXINFO Returns cgsix-specific information about the hardware. See
the definition of cg6_info in <sys/fbio.h> for more
information.

cgsix has registers and memory that may be mapped with mmap(2), using
the offsets defined in <sys/cg6reg.h>.

FILES
/dev/fbs/cgsix0

SEE ALSO
mmap(2), fbio(7I)
NAME  cgthree – 8-bit color memory frame buffer

SYNOPSIS  /dev/fbs/cgthree

DESCRIPTION  cgthree is a color memory frame buffer. It provides the standard frame
buffer interface as defined in fbio(7I).

FILES  /dev/fbs/cgthree[0-9]

SEE ALSO  mmap(2), fbio(7I)
NAME     cgtwo – color graphics interface  
SYNOPSIS /dev/cgtwo  
DESCRIPTION The cgtwo interface provides access to the color graphics controller board, which is normally supplied with a 19” 66 Hz non-interlaced color monitor. It provides the standard frame buffer interface as defined in fbio(7I).

The hardware consumes 4 megabytes of VME bus address space. The board starts at standard address 0x400000. The board must be configured for interrupt level 4.

FILES     /dev/cgtwo[0-9]  
SEE ALSO mmap(2), fbio(7I)
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 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; the
names of the raw files 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. However, I/O
requests to the 2K-byte CD-ROM drive must be a multiple of 2K bytes.
Otherwise, the driver returns an EINVAL error, too.

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.

Fdisk partition 0 is to access the entire disk and is generally used by the
fdisk(1M) program.

FILES

/dev/dsk/cndn[s|p]n block device (IDE)
/dev/rdsk/cndn[s|p]n raw device (IDE)

where:

<table>
<thead>
<tr>
<th>cn</th>
<th>controller n</th>
</tr>
</thead>
<tbody>
<tr>
<td>ln</td>
<td>lun n (0-7)</td>
</tr>
<tr>
<td>sn</td>
<td>UNIX system slice n (0-15)</td>
</tr>
<tr>
<td>pn</td>
<td>fdisk partition (0)</td>
</tr>
</tbody>
</table>

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:
<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO: `fdisk(1M)`, `mount(1M)`, `lseek(2)`, `read(2)`, `write(2)`, `readdir(3C)`, `scsi(4)`, `vfstab(4)`, `attributes(5)`, `dkio(7I)`
NAME
cnft – device driver for Compaq NIC

SYNOPSIS
/dev/cnft

DESCRIPTION
The cnft Ethernet driver is a multi-threaded, loadable, clonable, STREAMS GLD driver. This driver supports the following controllers:

<table>
<thead>
<tr>
<th>Controller</th>
<th>Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compaq NetFlex-3/EISA</td>
<td>• 10Base-T UTP Module</td>
</tr>
<tr>
<td></td>
<td>• 10/100Base-TX UTP Module</td>
</tr>
<tr>
<td></td>
<td>• 100VG-AnyLAN UTP Module</td>
</tr>
<tr>
<td></td>
<td>• 100Base-FX Module</td>
</tr>
<tr>
<td>Compaq NetFlex-3/PCI</td>
<td>• 10Base-T UTP Module</td>
</tr>
<tr>
<td></td>
<td>• 10/100Base-TX UTP Module</td>
</tr>
<tr>
<td></td>
<td>• 100VG-AnyLAN UTP Module</td>
</tr>
<tr>
<td></td>
<td>• 100Base-FX Module</td>
</tr>
<tr>
<td>Compaq Netelligent 10 T PCI UTP</td>
<td></td>
</tr>
<tr>
<td>Compaq Netelligent 10/100 TX PCI UTP</td>
<td></td>
</tr>
<tr>
<td>Compaq Dual Port NetFlex-3 10/100TX PCI UTP</td>
<td></td>
</tr>
<tr>
<td>.nf Compaq Integrated NetFlex-3 10/100T PCI</td>
<td>10/100T PCI with AUI on ProLiant 2500 and Professional Workstation 5000.</td>
</tr>
<tr>
<td>Compaq Integrated NIC on DeskPro</td>
<td>4000/6000 and ProLiant 800</td>
</tr>
</tbody>
</table>

Multiple controllers installed within the system are supported by the driver. The cnft driver provides basic support for these controllers. Functions include chip initialization, frame transmit and receive, multicast support, and error recovery and reporting and promiscuous mode support.

The cloning character-special device /dev/cnft is used to access all the above mentioned network controllers installed on the system.

The driver binary cnft and the configuration file cnft.conf must be present in /kernel/drv directory.

On Solaris 2.5, 2.5.1, and 2.6, for PCI controllers, the driver has to be added using the command

```
add_drv -i 'pciVID,DID'
```
where **VID** is the Vendor ID and **DID** is the Device ID of the PCI controller. Given below are the vendor ID and device ID of Compaq PCI NICs:

- e11,f130 NetFlex-3/P Controller
- e11,f150 NetFlex-3/P Controller (with TLAN 2.3)
- e11,ae32 Netelligent 10/100 TX PCI UTP Controller
- e11,ae34 Netelligent 10 T PCI UTP Controller
- e11,ae40 NetFlex-3 Dual Port 10/100TX PCI UTP
- e11,ae43 Integrated NetFlex-3 on ProLiant 2500 and Professional Workstation 5000
- e11,ae35 Integrated NIC on DeskPro 4000/6000 and ProLiant 800

For example, to add the Netelligent 10 T PCI UTP Controller, the command to be used is:

```
add_drv -i "pci11,ae34"
```

On Solaris 2.5/2.5.1/2.6, the NetFlex-3/E controller can be added by using the command

```
add_drv cnft
```

On Solaris 2.6 systems, an entry must be present in the master file for EISA NICs.

For example, an entry for both the EISA controllers will be as shown below:

```
CPQF120|CPQF140 cnft net all cnft.bef "NetFlex-3 EISA"
```

**CONFIGURATION**

The configuration file contains only the user defined properties.

The /kernel/drv/cnft.conf file supports the following options:

- **duplex_mode**
  - The *duplex_mode* can be selected using this property. This entry is optional and if not defined, *autosense* is taken as the default duplex mode.
  - The values are:
    - 0: Board autosenses the duplex mode.
    - 1: Half duplex mode.
    - 2: Full duplex mode.
max_tx_lsts
The maximum transmit lists for the controller. Every frame transmitted is described by a “list”. This value defines the maximum number of frames the driver can buffer before the controller actually transmits the frame over the media. This property is optional and a value of 16 is used by default.

max_rx_lsts
The maximum receive lists for the controller. Every frame received is described by a “list”. This value defines the maximum number of receive buffers provided to the controller by the driver. The controller will buffer as many frames before the driver picks them up. This property is optional and a value of 16 is used by default.

tx_threshold
The value of transmit threshold for the controller. This is the number of transmit frame complete (TX EOF) interrupts that must accumulate in the controller before it will generate an interrupt, thereby conserving interrupt overhead on the computer. This property is optional and a value of 2 is used by default.

media_speed
This property is used to force the media speed for the controller. It can be used to force a 10/100Base-TX interface to 10Mbps or 100Mbps operation. The values are:

0    Board autosenses the media speed.
10   Force 10Base-T operation.
100  Force 100Base-TX operation.

mediaconnector
This property is used by the driver to enable the AUI connector for the Integrated NetFlex-3 controller on ProLiant 2500 or the BNC connector for the Integrated NIC on DeskPro 4000/6000, ProLiant 800, and Professional Workstation 5000. The value is:

1    Use AUI Interface / Use BNC Interface
debug_flag
This property enables or disables the debug property of the driver. This is optional and by default it is disabled. The values are:

0 Disable the debug property.
1 Enable the debug property.

board_id
This property is used to support additional controller IDs. The format is:

0xVIDDID
where VID is the Vendor ID and DID the device ID.

FILES
/dev/cnft cnft character special device
/kernel/drv/cnft.conf configuration file of cnft driver
/sys/stropts.h
/sys/ethernet.h
/sys/gld.h

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO
attributes(5), dlpi(7P)
connld(7M)  STREAMS Modules

NAME
connld – line discipline for unique stream connections

SYNOPSIS
/dev/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.
STREAMS Modules

ENFILE

A file table entry could not be allocated.

SEE ALSO

creat(2), open(2), fattach(3C), streamio(7I)

STREAMS Programming Guide
console(7D) Devices

NAME
console – STREAMS-based console interface

SYNOPSIS
/dev/console

DESCRIPTION
The file /dev/console refers to the system console device.

SPARC
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)).

x86
By default the device is the “workstation console” device consisting of the workstation keyboard and display (see display(7D) and keyboard(7D)) acting in concert to emulate an ASCII terminal (see wscons(7D)).

In either architecture, 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
termios(3), ldterm(7M), termio(7I), wscons(7D)

x86 Only
display(7D), keyboard(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 workstation 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

cpqncr – low-level module for Compaq 32-Bit Fast-Wide SCSI-2 EISA/PCI (825) and Compaq Wide-Ultra SCSI PCI (875) Controllers

DESCRIPTION

The cpqncr module provides low-level interface routines between the common disk/tape I/O subsystem and the Compaq 825/875 SCSI (Small Computer System Interface) controllers.

The cpqncr module can be configured for disk and streaming tape support for one or more Compaq 825/875 controllers. Each 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

The driver attempts to initialize itself in accordance with the information found in the configuration file, cpqncr.conf. The relevant user configurable items in this file are as follows:

debug_flag

This property enables or disables driver debug messages. These messages are not displayed by default. Setting the value to 1 enables debug messages; setting it to 0 disables it.

alarm_msg_enable

This property enables alarm messages displayed for Storage System faults. Alarm messages are enabled by setting the value to 1 and disabled by setting it to 0. These messages are disabled by default.

tag_enable

This property enables or disables tag queueing support by the driver. Tagged Queueing is disabled by default. Tagged queueing is enabled by setting the value to 1 and disabled by setting the value to 0.

queue_depth

This property sets the number of active requests the driver can handle for a controller. The maximum and default value is 37 and the minimum value is 13. This can be decreased for supporting multiple controllers.

board_id

This property enables support for Compaq SCSI controllers other than Compaq 825/875 controllers. The board ID (Vendor and Device ID) must be specified for the driver to support the controller.
FILES
/kernel/drv/cpqncr.conf configuration file for the cpqncr driver

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO
driver.conf(4), attributes(5)
NAME

cpr – suspend and resume module

SYNOPSIS

/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.

The POWER key and the SHIFT+POWER keys on a type 5 keyboard access this module. Two utilities that may be installed on your system that will access this module are uadmin(1M) and uadmin(2).

The module performs the following actions when suspending the system. The signal SIGFREEZE is first sent to all user threads and then the threads are stopped. The system is brought down to a uni-processor mode for multi-processor systems. Dirty user pages are then swapped out to their backing storage device and all file systems are synchronized. All devices are made quiescent and system interrupts are disabled. To complete the system suspend, the kernel memory pages and remaining user pages are written to the root file system in a compressed form.

When the system is powered on again, essentially the reverse of the suspend procedure occurs. The kernel image is restored from the root file system by the bootstrapper /cprboot, interrupts and devices are restored to their previous state. Finally the user threads are rescheduled and SIGTHAW is broadcast to notify any interested processes of system resumption. Additional processors, if available, are restored and brought online. The system is now back to exactly the state prior to suspension.

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 these additional devices may not support cpr. In this case, the suspend will fail and an error message will be displayed to that effect. 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 cpr. 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 always 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.

The speed of suspend and resume operations can range from 15 seconds to several minutes, depending on the system speed, memory size, and load. The typical time is approximately one minute.

**FILES**

- /cprboot: special bootstrapper for cpr
- /.CPR: system state file

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWcpr</td>
</tr>
</tbody>
</table>

**SEE ALSO**

uadmin(1M), uadmin(2), attributes(5)

**NOTES**

For suspend/resume to work on multi-processor platforms, it must be able to control all CPUs. It is recommended that no MP tests (such as sundiag CPU tests) are running when suspend is initiated, because the suspend may be rejected if it cannot shut off all CPUs.

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.

**BUGS**

The signals SIGFREEZE and SIGTHAW are not properly implemented for the Solaris 2.4 release. This should only be a concern for specially customized applications that need to perform additional tasks at suspend or resume time.

In extremely rare occasions, the system may fail during the early stages of a resume operation. In this small window it is theoretically possible to be stuck in a loop that the system does not resume and it does not boot normally. If you are in such a loop, get to the prom ok prompt using the L1+A keys and enter the following command:

<ok> set-default boot-file
This command resets the system and with the next power-on the system will boot normally.
NAME

csa – low-level module for Compaq SMART Array Controller

DESCRIPTION

The csa module provides low-level interface routines between the common disk I/O subsystem and the Compaq SMART Array controllers. The csa module can be configured for disk support for one or more host adapter boards. Auto configuration code determines if the adapter is present at the configured address and what types of logical devices are configured on it.

CONFIGURATION

The driver attempts to initialize itself in accordance with the information found in the configuration file, /kernel/drv/csa.conf and with the information recorded in the EISA NVRAM. Each controller can support up to eight logical devices. The number and sizes of the logical devices are specified via the Compaq EISA Configuration Utility (ECU).

The user-configurable item in csa.conf is:

nccbs  The number of buffers which the driver should allocate to each controller board. The value specified should be between 16 and 255.

FILES

/kernel/drv/csa.conf  configuration file for csa

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO

attributes(5)
NAME  cvc – virtual console driver

DESCRIPTION  cvc is a STREAMS-based pseudodriver that supports the network console, which is called cvc on the host side and netcon on the SSP. cvc 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 the JTAG interface.

The cvc driver receives console input from cvcredir and passes it to the process associated with /dev/console.

NOTES  The cvc facility supercedes 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 the Enterprise 10000 system, which has no local keyboard or frame buffer.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Sun Enterprise 10000 servers only</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWcvc.u</td>
</tr>
</tbody>
</table>

SEE ALSO  cvcd(1M), attributes(5), console(7D), cvcredir(7D), wscons(7D), netcon(1M), netcon_server(1M) in the Sun Enterprise 10000 SSP Reference Manual.
NAME

cvcredir – virtual console redirection driver

DESCRIPTION

cvcredir, the virtual console redirection driver for the Enterprise 10000 server, is a STREAMS-based pseudodriver that works in conjunction with the cvc driver, cvc(7D), and the cvc daemon, cvcd(1M).

The cvcredir device is opened at start-of-day by the cvc daemon, cvcd. 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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Sun Enterprise 10000 servers only</td>
</tr>
<tr>
<td>Availability</td>
<td>SUNWcvc.u</td>
</tr>
</tbody>
</table>

SEE ALSO

cvcd(1M), attributes(5), console(7D), cvc(7D)

netcon(1M), netcon_server(1M) in the Sun Enterprise 10000 SSP Reference Manual
NAME
dbri – Dual Basic Rate ISDN and audio Interface

DESCRIPTION
The dbri device uses the T5900FC Dual Basic Rate ISDN Interface (DBRI) and Multimedia Codec chips to implement the audio device interface. This interface is described fully in the audio manual page.

Applications that open /dev/audio may use the AUDIO_GETDEV ioctl to determine which audio device is being used. The dbri driver will return the string "SUNW,dbri" in the name field of the audio_device structure. The version field will contain "e" and the config field will contain one of the following values: "isdn_b" on an ISDN B channel stream, "speakerbox" on a /dev/audio stream associated with a SpeakerBox, and lastly "onboard1" on a /dev/audio stream associated with the onboard Multimedia Codec.

The AUDIO_SETINFO ioctl controls device configuration parameters. When an application modifies the record.buffer_size field using the AUDIO_SETINFO ioctl, the driver will constrain it to be non-zero and a multiple of 16 bytes, up to a maximum of 8176 bytes.

Audio Interfaces
The SpeakerBox audio peripheral is available for connection to the SpeakerBox Interface (SBI) port of most dbri equipped systems and provides an integral monaural speaker as well as stereo line out, stereo line in, stereo headphone, and monaural microphone connections. The headset output level is adequate to power most headphones, but may be too low for some external speakers. Powered speakers or an external amplifier may be used with both the headphone and line out ports.

SPARCstation LX systems have the Multimedia Codec integrated onto the CPU board of the machine thus giving users the option of using it or using a SpeakerBox plugged into the AUI/Audio port on the back panel. When using the "onboard" Codec, the microphone and headphone ports are located on the system back panel - there are no Line In or Line Out ports available for this configuration. In addition, the headphone and microphone ports do not have the input detection circuitry to determine whether or not there is currently headphones or a microphone plugged in. If a SpeakerBox is plugged in when the machine is first rebooted and reconfigured, or upon the first access of the audio device, it will be used, otherwise the onboard Codec will be used.

The Sun Microphone is recommended for normal desktop audio recording. When the Sun Microphone is used in conjunction with the SpeakerBox, the microphone battery is bypassed. Other audio sources may be recorded by connecting their line output to the SpeakerBox line input (audio sources may also be connected from their headphone output if the volume is adjusted properly).
ISDN Interfaces

The DBRI controller offers two Basic Rate ISDN (BRI) interfaces. One is a BRI Terminal Equipment (TE) interface and the other is a BRI Network Termination (NT) interface.

The NT connector is switched by a relay so that when system power is not available or when software is not accessing the NT port, the TE and NT connectors are electrically connected and devices plugged into the NT port will be on the same BRI passive bus.

Audio Data Formats for the Multimedia Codec/SpeakerBox

The dbri device supports the audio formats listed in the following table. When the device is open for simultaneous play and record, the input and output data formats must match.

<table>
<thead>
<tr>
<th>Sample Rate</th>
<th>Encoding</th>
<th>Precision</th>
<th>Channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>8000 Hz</td>
<td>mu-law (as in the Greek letter mu) or A-law</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>9600 Hz</td>
<td>mu-law or A-law</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>11025 Hz</td>
<td>mu-law or A-law</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>16000 Hz</td>
<td>mu-law or A-law</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>18900 Hz</td>
<td>mu-law or A-law</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>22050 Hz</td>
<td>mu-law or A-law</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>32000 Hz</td>
<td>mu-law or A-law</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>37800 Hz</td>
<td>mu-law or A-law</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>44100 Hz</td>
<td>mu-law or A-law</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>48000 Hz</td>
<td>mu-law or A-law</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>8000 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
</tr>
<tr>
<td>9600 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
</tr>
<tr>
<td>11025 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
</tr>
<tr>
<td>16000 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
</tr>
<tr>
<td>18900 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
</tr>
<tr>
<td>22050 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
</tr>
<tr>
<td>32000 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
</tr>
<tr>
<td>37800 Hz</td>
<td>linear</td>
<td>16</td>
<td>1 or 2</td>
</tr>
</tbody>
</table>
ISDN channels implement a subset of audio semantics. The preferred ioctls for querying or setting the format of a BRI channel are `ISDN_GET_FORMAT`, `ISDN_SET_FORMAT`, and `ISDN_SET_CHANNEL`. In particular, there is no audio format described in `audio(7I)` that covers HDLC or transparent data. The `dbri` driver maps HDLC and transparent data to `AUDIO_ENCODING_NONE`. ISDN D-channels are always configured for HDLC encoding of data. The programmer should interpret an encoding value of `AUDIO_ENCODING_NONE` as an indication that the `fd` is not being used to transfer audio data.

B-channels can be configured for mu-law (as in the Greek letter mu), A-law, or HDLC encoding of data. The mu-law and A-law formats are always at 8000 Hz, 8-bit, mono. Although a BRI H-channel is actually 16 bits wide at the physical layer and the 16-bit sample occurs at 8 kHz, the HDLC encoding always presents the data in 8-bit quantities. Therefore, 56 bit-per-second (bps), 64 bps, and 128 bps formats are all presented to the programmer as 8-bit wide, mono, `AUDIO_ENCODING_NONE` format streams at different sample rates. A line rate of 56kbps results in a 8-bit sample rate of 7000 Hz. If the bit stuffing and un-stuffing of HDLC were taken into account, the data rate would be slightly less.

For the sake of compatibility, `AUDIO_GETINFO` will return one of the following on a ISDN channel:

<table>
<thead>
<tr>
<th>Sample Rate</th>
<th>Encoding</th>
<th>Precision</th>
<th>Channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>8000 Hz</td>
<td>mu-law or A-law</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>AUDIO_ENCODING_NONE</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

`ISDN_GET_FORMAT` will return one of the following for an ISDN channel:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Sample Rate</th>
<th>Encoding</th>
<th>Precision</th>
<th># Ch</th>
<th>Available on</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDLC</td>
<td>2000 Hz</td>
<td>NONE</td>
<td>8</td>
<td>1</td>
<td>D</td>
</tr>
<tr>
<td>HDLC</td>
<td>7000 Hz</td>
<td>NONE</td>
<td>8</td>
<td>1</td>
<td>B1,B2</td>
</tr>
<tr>
<td>HDLC</td>
<td>8000 Hz</td>
<td>NONE</td>
<td>8</td>
<td>1</td>
<td>B1,B2</td>
</tr>
</tbody>
</table>
dbri(7D) Devices

<table>
<thead>
<tr>
<th>HDLC</th>
<th>16000 Hz</th>
<th>NONE</th>
<th>8</th>
<th>1</th>
<th>B1,B2</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANS</td>
<td>8000 Hz</td>
<td>mu-law</td>
<td>8</td>
<td>1</td>
<td>B1,B2</td>
</tr>
<tr>
<td>TRANS</td>
<td>8000 Hz</td>
<td>A-law</td>
<td>8</td>
<td>1</td>
<td>B1,B2</td>
</tr>
<tr>
<td>TRANS</td>
<td>8000 Hz</td>
<td>NONE</td>
<td>8</td>
<td>1</td>
<td>B1,B2</td>
</tr>
<tr>
<td>TRANS</td>
<td>8000 Hz</td>
<td>NONE</td>
<td>16</td>
<td>1</td>
<td>B1 only</td>
</tr>
</tbody>
</table>

In the previous table, HDLC = ISDN_MODE_HDLC, TRANS = ISDN_MODE_TRANSPARENT.

Audio Ports

Audio ports are not relevant to ISDN D or B channels.

The `record.avail_ports` and `play.avail_ports` fields of the `audio_info` structure report the available input and output ports. The `dbri` device supports two input ports, selected by setting the `record.port` field to either `AUDIO_MICROPHONE` or `AUDIO_LINE_IN`. The `play.port` field may be set to any combination of `AUDIO_SPEAKER`, `AUDIO_HEADPHONE`, and `AUDIO_LINE_OUT` by OR’ing the desired port names together. As noted above, when using the onboard Multimedia Codec on the SPARCstation LX, the Line In and Line Out ports are not available.

Sample Granularity

Since the `dbri` device manipulates buffers of audio data, at any given time the reported input and output sample counts will vary from the actual sample count by no more than the size of the buffers it is transferring. Programs should, in general, not rely on absolute accuracy of the `play.samples` and `record.samples` fields of the `audio_info` structure.

Audio Status Change Notification

As described in `audio(7I)`, it is possible to request asynchronous notification of changes in the state of an audio device. The DBRI driver extends this to the ISDN B channels by sending the signal up the data channel instead of the control channel. Asynchronous notification of events on a B-channel only occurs when the channel is in a transparent data mode. When the channel is in HDLC mode, no such notification will take place.

ERRORS

In addition to the errors described in `audio(7I)`, an `open()` will fail if:

- **ENODEV**: The driver is unable to communicate with the SpeakerBox, possibly because it is currently not plugged in.

FILES

The physical device names are very system dependent and are rarely used by programmers. For example:

`/devices/sbus@1,f8000000/SUNW,DBRIe@1,10000:te,b2`.

The programmer should instead use the generic device names listed below:

`/dev/audio` - symlink to the system’s primary audio device, not necessarily a dbri based audio device

95 SunOS 5.7 Last modified 1 Jan 1997
Devices

```
/dev/audioctl - control device for the above audio device
/dev/sound/0* - represents the first audio device on the system and is not necessarily based on dbri or SpeakerBox
/dev/sound/0 - first audio device in the system
/dev/sound/0ctl - audio control for above device
/dev/isdn/0/*/ - represents the first ISDN device on the system and any associated interfaces. This device is not necessarily based on dbri.
/dev/isdn/0/te/mgt - TE management device
/dev/isdn/0/te/d - TE D-channel
/dev/isdn/0/te/b1 - TE B1-channel
/dev/isdn/0/te/b2 - TE B2-channel
/dev/isdn/0/nt/mgt - NT management device
/dev/isdn/0/nt/d - NT D-channel
/dev/isdn/0/nt/b1 - NT B1-channel
/dev/isdn/0/nt/b2 - NT B2-channel
/dev/isdn/0/aux/0 - SpeakerBox or onboard Multimedia Codec
/dev/isdn/0/aux/0ctl - Control device for SpeakerBox or onboard Multimedia Codec
/usr/demo/SOUND - audio demonstration programs and other files
```

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>SPARC</td>
</tr>
</tbody>
</table>

The DBRI Multimedia Codec, and SpeakerBox are available on SPARCstation 10 and LX systems.

SPARCstation 10SX and SPARCstation 20 systems have the Multimedia Codec integrated onto the CPU board of the machine.

Last modified 1 Jan 1997

SunOS 5.7

96
This hardware may or may not be available on future systems from Sun Microsystems Computer Corporation.

There are new configurations for the SX10SX and Gypsy machines. The SS10BSX looks like a speakerbox but does not have auto-detection of the Headphone and Microphone ports. The Gypsy claims to be "onboard" but does have line in and line out ports.

SEE ALSO

ioctl(2), attributes(5), audio(7I), isdnio(7I), streamio(7I)

AT&T Microelectronics data sheet for the T5900FC Sun Dual Basic Rate ISDN Interface.

Crystal Semiconductor, Inc., data sheet for the CS4215 16-Bit, 48 kHz, Multimedia Audio Codec Publication number DS76PP5.

NOTES

Due to hardware restrictions, it is impossible to reduce the record gain to 0. A valid input signal is still received at the lowest gain setting the Multimedia Codec allows. For security reasons, the dbri driver disallows a record gain value of 0. This is to provide feedback to the user that such a setting is not possible and that a valid input signal is still being received. An attempt to set the record gain to 0 will result in the lowest possible non-zero gain. The audio_info structure will be updated with this value when the AUDIO_SETINFO ioctl returns.

BUGS

When a DBRI channel associated with the SpeakerBox Interface underruns, DBRI may not always repeat the last sample but instead could repeat more than one sample. This behavior can result in a tone being generated by an audio device connected to the SBI port.

Monitor STREAMs connected to a B1 channel on either the TE or NT interface do not work because of a DBRI hardware problem. The device driver disallows the creation of such monitors.
NAME
devinfo – device information driver

DESCRIPTION
The devinfo driver is a private mechanism used by the libdevinfo 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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stability Level</td>
<td>Private</td>
</tr>
</tbody>
</table>

SEE ALSO
libdevinfo(3), libdevinfo(4), attributes(5)

Writing Device Drivers

Last modified 9 Jan 1998
display (7D)  Devices

NAME   display – system console display

DESCRIPTION  display is a component of the kd driver, which is comprised of the display and keyboard drivers.

Solaris for x86 normally uses a windowed environment. The character-based display facilities offered by the display section of the kd driver are supposed to be used only until the windowing system takes over. Currently, any VGA adapter can be used to boot the system, but the windowing server requires an SVGA or 8514 adapter.

See the supported hardware list in the for the full list of tested adapters.

FILES   /dev/console

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO  attributes (5), console (7D), keyboard (7D)

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 type of the controller and attributes about how bad-block processing is 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 */

}
dkio(7I)  Ioctl Requests

```c
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 otherwise */
#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_MD21          8
```
# 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 /* metadisk (virtual-disk) */
# define DKC_INTEL82077 19 /* 82077 floppy disk */
# define DKC_DIRECT 20 /* Intel direct attached */
# define DKC_PCMCIA_MEM 21 /* PCMCIA memory disk-like */
# 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 forwarding */
#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 hex */
  /* digits */
#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 VTOC (volume table of contents).

DKIOCSVTOC The argument is a pointer to a vtoc structure (described below). This ioctl() changes the VTOC associated with the device.

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. Multiply the number of heads by the number of sectors per track to compute the number of sectors per cylinder.

```c
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 */
    /* partition */
    /* timestamp */
    /* (unsupported) */
    char v_asciilabel[LEN_DKL_ASCIILABEL]; /* 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 */

DKIOCEJECT
This ioctl() requests the disk drive to eject its disk, if that drive supports removable media.

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 {

The argument to this ioctl() is an integer. After successful completion, this ioctl() will set that integer to a non-zero value if the drive in question has removable media. If the media is not removable that integer will be set to 0.

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.

```c
enum dkio_state {
    DKIO_NONE, /* Return disk’s current state */
    DKIO_EJECTED, /* Disk state is ‘ejected’ */
    DKIO_INSERTED /* Disk state is ‘inserted’ */
};
```

This ioctl() requests the disk drive to lock the door, for those devices with removable media.

This ioctl() requests the disk drive to unlock the door, for those devices with removable media.

The following ioctl() requests set and/or retrieve the current disk controller, partitions, or geometry information on x86 architectures:

- **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.

```c
/* Definition of a disk’s geometry
*/
struct dk_geom {

    unsigned short dkg_ncyl;    /* # of data */
    /* cylinders */

    unsigned short 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_intr1v;  /* interleave factor */

    unsigned short dkg_obs2;    /* obsolete */

    unsigned short dkg_obs3;    /* obsolete */

    unsigned short dkg_apc;     /* alternates per */
    /* cyl (SCSI only) */

    unsigned short dkg_rpm;     /* revolutions per */
    /* min */

    unsigned short dkg_pcy1;    /* # of physical */
    /* cylinders */

    unsigned short dkg_write_reinstruct;    /* # sectors to */
    /* skip, writes */

    unsigned short dkg_read_reinstruct;     /* # sectors to */
    /* skip, reads */
}

Last modified 9 Feb 1998

SunOS 5.7
DKIOCADDDBAD

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. DKIOCADDDBAD can only be used for software remapping on IDE drives; SCSI drives use hardware remapping of alternate sectors.

SEE ALSO

format(1M), ioctl(2), cdio(7I), fdio(7I), hdio(7I), sd(7D), xd(7D), xy(7D)

x86 Only

addbadsec(1M), cmdk(7D)
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.

For details on this interface refer to the <sys/dlpi.h> header and to the
STREAMS DLPI Specification, 800-6915-01.

FILES
Files in or under /dev.

SEE ALSO
ie(7D), le(7D)
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.

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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO attributes(5), dlpi(7P), gld(7D) streamio(7I)

Writing Device Drivers STREAMS Programming Guide Network Interfaces Programmer's Guide
NAME

SYNOPSIS
ISA, EISA:

dpt@ioaddr, 0

DESCRIPTION
The dpt module provides low-level interface routines between the common disk/tape I/O subsystem and the DPT ISA bus master 2011 and 2021 Small Computer System Interface (SCSI) controllers, the DPT EISA 2012, 2022, and 2122 SCSI controllers, the DPT PCI 2024 and 2124 SCSI controllers, and the DPT SCSI RAID controllers: 3021 (ISA), 3222 (EISA), and 3224 (PCI).

The dpt 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 it. If a memory cache module is installed on the DPT board, this cache will be flushed to disk by the dpt driver module when the system is shut down by the system administrator.

FILES
/platform/i86pc/kernel/drv/dpt.conf
Configuration file for the dpt driver. There are no user-configurable options in this file.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO sysbus(4), attributes(5)
ecpp – IEEE 1284 ecp, nibble and centronics compatible parallel port driver

#include <sys/types.h>
#include <fcntl.h>
#include <sys/ecppio.h>

The `ecpp` driver provides a bi-directional interface to IEEE 1284 compliant devices. The driver will operate in Centronics mode for non-IEEE 1284 compliant devices. An IEEE 1284 compliant peripheral device must operate at least in Compatibility mode and Nibble mode. The `ecpp` driver supports Compatibility, Nibble and ECP modes of operation as defined by IEEE 1284. Centronics and Compatibility modes of operation have identical physical characteristics. However, non-IEEE 1284 compliant devices will be logically defined as `ECPP_CENTRONICS`. IEEE 1284 devices that are in a similar mode will be logically defined as `ECPP_COMPAT_MODE`. `ECPP_COMPAT_MODE` operates in conjunction with `ECPP_NIBBLE_MODE`. The `ecpp` driver is an exclusive-use device. If the device has already been opened, subsequent opens fail with EBUSY.

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 60 seconds. The driver sets the mode variable according to the following algorithm: The driver initially attempts to negotiate the device into ECP mode. If this fails, the driver will attempt to negotiate into Nibble mode. If Nibble mode negotiation fails, the driver will operate in Centronics mode. The application may attempt to negotiate the device into a specific mode or set the `write_timeout` values through the `ECPP_IOC_SETPARMS ioctl(2)` call. In order for the negotiation to be successful, both the host workstation and the peripheral must support the requested mode.

The preferred mode of operation of an IEEE 1284 device is the bi-directional ECP mode. Nibble mode is a unidirectional backchannel mode. It utilizes a PIO method of transfer and consequently, is inefficient. For devices that primarily receive data from the workstation, such as printers, Nibble operation will have limited impact to system performance. Nibble mode should not be used for devices such as a scanner, that primarily send data to the workstation. Forward transfers under all modes are conducted through a DMA method of transfer.

The default timeout for the `ecpp` device driver may be changed by adding the following line to the `/etc/system` file:

```
set ecpp:ecpp_def_timeout = value
```

See `system(4)` for more details.
ecpp 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(2)` will return when all the data has been successfully transferred to the device.

**Write Operation**

`write()` 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 `ECPPIOC_GETERR ioctl()` call. The captured status information will be overwritten each time an attempted transfer or a `ECPPIOC_TESTIO ioctl()` occurs.

Intelligent IEEE 1284 compliant devices, such as Postscript printers, return error information through a backchannel. This data may be retrieved with the `read(2)` call.

**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 the port that has no data currently available, `read()` returns 0 if `O_NDELAY` is set. If `O_NONBLOCK` is set, `read()` returns -1 and sets `errno` to `EAGAIN`. If `O_NDELAY` and `O_NONBLOCK` are clear, `read()` blocks until data become available.

**IOCTLS**

The following `ioctl(2)` calls are supported:

- **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.
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.

Transfer Parameters Structure
This structure is defined in <sys/ecppio.h>.

```
struct ecpp_transfer_parms {
    int write_timeout;
    int mode;
};
```

The write_timeout field is set to ECPP_W_TIMEOUT_DEFAULT. 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. Any other values are out of range. The mode field reflects the IEEE 1284 mode that the parallel port is currently configured to. The mode may be set to only one of the following bit values.

```
#define ECPP_CENTRONICS 0x1
#define ECPP_COMPAT_MODE 0x2
#define ECPP_NIBBLE_MODE 0x3
#define RESERVED 0x4
#define ECPP_FAILURE_MODE 0x5
```
This command may set the mode value to ECPP_CENTRONICS, ECPP_COMPAT_MODE, or ECPP_NIBBLE_MODE. All other values are not valid. If the requested mode is not supported, ECPPIOC_SETPARMS will return EPROTONOSUPPORT. Under this circumstance, ECPPIOC_GETPARMS will return to its original mode. If a non-recoverable IEEE 1284 error occurs, the driver will be set to ECPP_FAILURE_MODE. For instance, if the port is not capable of returning to its original mode, ECPPIOC_GETPARMS will return ECPP_FAILURE_MODE.

**BPPIOC_TESTIO**

Tests the transfer readiness of ECPP_CENTRONICS or ECPP_COMPAT_MODE devices. If the current mode of the port is ECPP_CENTRONICS or ECPP_COMPAT_MODE, this command determines if write(2) would succeed. If it is not one of these modes, EINVAL is returned. BPPIOC_TESTIO determines if a write(2) would succeed by checking the open flag and status pins. If any of the status pins are set, a transfer would fail. If a transfer would succeed, zero is returned. If a transfer would fail, -1 is returned, and 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. Note that 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(7). However, bus_error is not used in this interface.

**BPPIOC_GETERR**

Get last error status.

The argument is a pointer to a struct bpp_error_status. This structure is described below. This structure indicates the status of all the appropriate status bits at the time of the most recent error condition during a write() call, or the status of the bits at the most recent BPPIOC_TESTIO ioctl() call.
The `timeout_occurred` value is set when a timeout occurs during `write()`. `bus_error` is not used in this interface.

`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. As defined in the IEEE 1284 Specification, status signals do not represent the error status of ECP devices. Error information is formatted by a printer specific protocol such as PostScript, and is returned through the backchannel.

Error Status Structure

```c
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.
                      */
};
/* pin_status values */
#define BPP_ERR_ERR 0x01 /* nErr=0 */
#define BPP_SLCT_ERR 0x02 /* Select=1 */
#define BPP_PE_ERR 0x04 /* PE =1 */
#define BPP_BUSY_ERR 0x40 /* Busy = 1 */
```

**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** `ECPPIOC_SETPARMS ioctl()` is attempted with an out of range value in the `ecpp_transfer_parms` structure. A `ECPPIOC_SETREGS`
An 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 from the vd driver during modload(1M), modunload(1M).

EIO The driver encountered a bus error when attempting an access.

A read or write does 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.

FILES
/dev/ecpp0 1284 compatible and ecp mode parallel port device.

SEE ALSO ioctl(2), read(2), write(2), system(4), streamio(7I)
NAME

eepro – Intel EtherExpress-Pro Ethernet device driver

SYNOPSIS

/dev/eepro

DESCRIPTION

The eepro Ethernet driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface, dlpi(7P), over Intel EtherExpress-Pro Ethernet controllers. The EtherExpress-Pro Ethernet adapter is based on the Intel 82595TX high integration controller. Multiple EtherExpress-Pro controllers installed within the system are supported by the driver.

The eepro driver provides basic support for the EtherExpress-Pro hardware. Functions including chip initialization, frame transmit and receive, multicast and “promiscuous” support, and error recovery and reporting. It also supports an ioctl to perform a time domain reflectometry test (i.e. detect open or short circuits on the link). Refer to IOCTLs below.

APPLICATION PROGRAMMING INTERFACE

The cloning, character-special device /dev/eepro is used to access all EtherExpress-Pro devices installed within the system.

The eepro driver is dependent on /kernel/misc/gld, a loadable kernel module that provides the eepro 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 - defined in <sys/ethernet.h>).
- The minimum SDU is 0. The driver will pad to the mandatory 60-byte minimum packet size.
- The dlsap address length is 8.
- The Media Access Control (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

Auto-detect of the media type is supported and the board need not be explicitly configured for which media connector it is using. It is important to ensure that there are no conflicts between the board’s I/O port or IRQ level and other hardware installed in the system.
FILES
/dev/eepro  eepro character special device
/kernel/drv/eepro.conf  configuration file of eepro driver

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO  attributes(5), dlpi(7P), gld(7D)
NAME

eha – low-level module for Adaptec 174x EISA host bus adapter

DESCRIPTION

The eha module provides low-level interface routines between the common disk/tape IO subsystem and the Adaptec EISA 174x SCSI (Small Computer System Interface) controllers. The eha 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 it.

Board Configuration and Auto Configuration

The driver attempts to initialize itself in accordance with the information found in the configuration file, /kernel/drv/eha.conf. The only relevant user configurable item in this file is:

"io address "reg=0x1000,0,0" "iaddr=0x1000"

The I/O address is 0x1000 times the EISA slot number. Hence, slot 1 is address 0x1000 and slot 8 is 0x8000.

Prior to installation, the 174x controller must be put into enhanced mode with the EISA configuration utility run under MS-DOS.

The default listing of the configuration file is as follows:

```
# primary controller [Settings for CD-ROM installation]
#
name="eha" class="sysbus" reg=0x1000,0,0
  iaddr=0x1000;
# another controller example
#
name="eha" class="sysbus" reg=0x2000,0,0
  iaddr=0x2000;
#
```

To speed boot, parameters in the configuration file may be commented out with a "#" in the first column for controllers that are not installed.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO

attributes(5)

Last modified 1 Jan 1997

SunOS 5.7

122
NAME elx – 3COM EtherLink III Ethernet device driver

SYNOPSIS
#include <sys/stropts.h>
#include <sys/ethernet.h>
#include <sys/dlpi.h>
#include <sys/gld.h>

DESCRIPTION
The elx Ethernet driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface, dlpi(7P), over the following 3COM ETHERLINK III Ethernet controllers. For x86 based systems: 3C509, 3C509B, 3C579 and 3C59x controllers. Multiple EtherLink III controllers installed within the system are supported by the driver. The elx driver provides basic support for the EtherLink III hardware. Functions include chip initialization, frame transmit and receive, multicast and “promiscuous” support, and error recovery and reporting.

The cloning, character-special device /dev/elx is used to access all EtherLink III devices installed within the system.

The elx driver is dependent on /kernel/misc/gld, a loadable kernel module that provides the elx 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:
The maximum SDU is 1500 (ETHERMTU).
The minimum SDU is 0. The driver will pad to the mandatory 60-octet minimum packet size.
The dl 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 broadcast address value is Ethernet/IEEE broadcast address (FF:FF:FF:FF:FF:FF).

FILES
/dev/elx special character device
/platform/i86pc/kernel/drv/elx.conf configuration file for elx driver

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

123 SunOS 5.7 Last modified 24 Feb 1998
### Devices

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

**SEE ALSO**

attributes(5), dlpi(7P), gld(7D)

Last modified 24 Feb 1998
SunOS 5.7
NAME
esa - low-level module for Adaptec 7770 based SCSI controllers

DESCRIPTION
The esa 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-7770 SCSI chip. These controllers include the
Adaptec 2740 and 2840, as well as motherboards with an embedded AIC-7770
SCSI chip.

The esa 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. Each
controller may support one or two SCSI busses, depending on the
manufacturer's implementation.

FILES
/kernel/drv/esa.conf configuration file for the esa driver. There are no
user-configurable options in this file.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO
attributes(5)
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/sd83,0:a,raw
```
The register property values can be determined from `prtconf`(1M) output
(−v option):

```
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 (ie. 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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>SBus-based systems with esp-based</td>
</tr>
<tr>
<td></td>
<td>SCSI port and SSHA, SBE/S, FSBE/S,</td>
</tr>
<tr>
<td></td>
<td>and DSBE/S SBus SCSI Host Adapter</td>
</tr>
<tr>
<td></td>
<td>options</td>
</tr>
</tbody>
</table>

---

**SEE ALSO**

`prtconf`(1M), `driver.conf`(4), `attributes`(5), `fas`(7D),
`scci_abort`(9F), `scci_hba_attach`(9F), `scci_ifgetcap`(9F),
`scci_reset`(9F), `scci_sync_pkt`(9F), `scci_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

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.
Devices esp(7D)

<message name=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.
Register Specifications:
Bus Type=0x2, Address=0x81000, Size=10

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 <0x000000a>.
  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="/iommu8f,e0000000/sbus8f,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.

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.
sscsi-options specified per target ID have the highest precedence, followed by sscsi-options per device type. Global fas sscsi-options (effecting all instances) per bus have the lowest precedence.

The system needs to be rebooted before the specified sscsi-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 sscsi-options and a capability, the value set in sscsi-options prevails. Only whom != 0 is supported in the sscsi_ifsetcap(9F) call.

Refer to sscsi_ifsetcap(9F) and sscsi_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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Limited to Sparc SBus-based systems with FAS366-based SCSI port and SunSWIFT SBus SCSI Host Adapter/Fast Ethernet option.</td>
</tr>
</tbody>
</table>
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

OpenBoot Command Reference

ANSI Small Computer System Interface-2 (SCSI-2)

FAS366 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 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 allocate soft state

Cannot alloc dma handle

Cannot alloc cmd area

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

Target `<id>` reverting to async. mode

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 <0x0000000b>
   name <scsi-tag-age-limit> length <4>
      value <0x00000002>
   name <scsi-reset-delay> length <4>
      value <0x000000bb8>

Register Specifications:
   Bus Type=0x3, Address=0x8800000, Size=10
   Bus Type=0x3, Address=0x8810000, Size=40
Interrupt Specifications:
   Interrupt Priority=0x35 (ipl 5)
## 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 `FBIOPUTCMAP1` and `FBIOGETCMAP1` `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.

`FBIMONINFO` returns a `mon_info` structure which contains information about the monitor attached to the framebuffer, if available.

`FBIOSCURSOR`, `FBIOSCURSOR`, `FBIOSCURPOS` and `FBIOSCURPOS` are used to control the hardware cursor for those framebuffers that have this feature. `FBIOSCURMAX` returns the maximum sized cursor supported by the framebuffer. Attempts to create a cursor larger than this will fail.

Finally `FBIOSDEVINFO` and `FBIOSDEVINFO` are used to transfer variable-length, device-specific information into and out of framebuffers.
SEE ALSO

ioctl(2), mmap(2), bwtwo(7D), cgeight(7D), cgfour(7D), cgsix(7D), cgthree(7D), cgtwo(7D)

BUGS

The FBIOSATTR and FBIOGATTR ioctl() requests are only supported by frame buffers which emulate older frame buffer types. For example, cgfour(7D) frame buffers emulate bwtwo(7D) frame buffers. If a frame buffer is emulating 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
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 driver provides the interfaces to the floppy disks using the Intel 82072 on sun4c systems and the Intel 82077 on sun4m systems.

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:

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>All cylinders except the last</td>
</tr>
<tr>
<td>b</td>
<td>Only the last cylinder</td>
</tr>
<tr>
<td>c</td>
<td>Entire diskette</td>
</tr>
</tbody>
</table>

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 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" 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 many words are transmitted.

3.5” Diskettes
For 3.5” double-sided diskettes, the following densities are supported:

<table>
<thead>
<tr>
<th>Density</th>
<th>Cylinders</th>
<th>Sectors per track</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.7 Mbyte</td>
<td>80</td>
<td>21</td>
<td>1.7 Mbyte</td>
</tr>
<tr>
<td>high density</td>
<td>80</td>
<td>18</td>
<td>1.44 Mbyte</td>
</tr>
</tbody>
</table>

Last modified 24 Feb 1998
### Devices

<table>
<thead>
<tr>
<th>Density Type</th>
<th>Cylinder Count</th>
<th>sectors per track</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double Density</td>
<td>80</td>
<td>9</td>
<td>720 Kbyte</td>
</tr>
<tr>
<td>Medium Density</td>
<td>77</td>
<td>8</td>
<td>1.2 Mbyte</td>
</tr>
<tr>
<td><strong>x86</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extended Density</td>
<td>80</td>
<td>36</td>
<td>2.88 Mbyte</td>
</tr>
<tr>
<td>1.7 Mbyte Density</td>
<td>80</td>
<td>21</td>
<td>1.7 Mbyte</td>
</tr>
<tr>
<td>High Density</td>
<td>80</td>
<td>18</td>
<td>1.44 Mbyte</td>
</tr>
<tr>
<td>Double Density</td>
<td>80</td>
<td>9</td>
<td>760 Kbyte</td>
</tr>
</tbody>
</table>

### 5.25" Diskettes

For 5.25" double-sided diskettes, the following densities are supported:

<table>
<thead>
<tr>
<th>Density Type</th>
<th>Cylinder Count</th>
<th>sectors per track</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>High Density</td>
<td>80</td>
<td>15</td>
<td>1.2 Mbyte</td>
</tr>
<tr>
<td>Double Density</td>
<td>40</td>
<td>9</td>
<td>360 Kbyte</td>
</tr>
<tr>
<td>Double Density</td>
<td>40</td>
<td>8</td>
<td>320 Kbyte</td>
</tr>
<tr>
<td>Quad Density</td>
<td>80</td>
<td>9</td>
<td>720 Kbyte</td>
</tr>
<tr>
<td>Double Density</td>
<td>40</td>
<td>16</td>
<td>256 bytes</td>
</tr>
<tr>
<td>Double Density</td>
<td>40</td>
<td>4</td>
<td>1024 bytes</td>
</tr>
</tbody>
</table>

### 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
Devices

fd(7D)

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/sun4c/kernel/drv/fd driver module
/platform/sun4m/kernel/drv/fd driver module
/platform/sun4u/kernel/drv/fd driver module
/usr/include/sys/fdreg.h structs and definitions for Intel 82072 and 82077 controllers

Last modified 24 Feb 1998

SunOS 5.7

150
/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
/vol/dev/diskette0  directory containing volume management character device file
/vol/dev/rdiskette0  directory containing the volume management raw character device file
/vol/dev/aliases/floppy0  symbolic link to the entry in /vol/dev/rdiskette0

/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
/vol/dev/diskette0 directory containing volume
management character device file
/vol/dev/rdiskette0 directory containing the volume
management raw character device file
/vol/dev/aliases/floppy0 symbolic link to the entry in
/vol/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
/vol/dev/diskette1 directory containing volume
management character device file
/vol/dev/rdiskette1 directory containing the volume
management raw character device file
/vol/dev/aliases/floppy1 symbolic link to the entry in
/vol/dev/rdiskette1

SEE ALSO fdformat(1), dd(1M), drvconfig(1M), vold(1M), read(2), write(2),
driver.conf(4), dkio(7I) fdio(7I)

DIAGNOSTICS 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:

Last modified 24 Feb 1998 SunOS 5.7 152
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

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.

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.

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)` to or from a 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 (Intel Platform Edition), the configuration of the floppy drives is specified in CMOS configuration memory. Use the BIOS setup program or an EISA configuration 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 (Intel Platform 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 that for x86 based systems, use FDGC_DETECTED (which is available only on x86 based systems) instead of FDGC_HISTORY.

#define FDGC_HISTORY 0x01 /* disk has changed since last call */
#define FDGC_CURRENT 0x02 /* current state of disk change */
#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:

fd_char
field value
fdc_medium 0
fdc_transfer_rate 500
fdc_ncyl 80
fdc_nhead 2
fdc_sec_size 512
fdc_secptrack 18

Last modified 13 Feb 1997		SunOS 5.7	154
ioctl Requests

/* Floppy characteristics */
struct fd_char {
    uchar_t fdc_medium; /* equals 1 if medium type */
    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 86 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 reducing 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.
SPARC: This ioctl() requests the floppy drive to eject the diskette.

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_TR commands are currently available.

```c
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.

```c
/* 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.

```c
/*
 * 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.

```c
/*
 * 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 ioctl's 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
flashpt – low-level module for Mylex/BusLogic host bus adapters

SYNOPSIS
pci104b,8130@d

DESCRIPTION
The flashpt module provides low-level interface routines between the common disk/tape I/O subsystem and the BusLogic FlashPoint Ultra SCSI (Small Computer System Interface) controllers. The flashpt module can be configured for disk and streaming tape support for one or more host bus 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 determines what types of devices are attached to the adapter.

Supported BusLogic Adapters
The following table describes the BusLogic host adapters supported by the flashpt module.

<table>
<thead>
<tr>
<th>MODEL</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>FlashPoint LT</td>
<td>PCI Ultra SCSI adapter</td>
</tr>
<tr>
<td>FlashPoint LW</td>
<td>PCI Ultra &amp; Wide SCSI adapter</td>
</tr>
<tr>
<td>FlashPoint DL</td>
<td>PCI Dual Channel Ultra SCSI adapter</td>
</tr>
<tr>
<td>FlashPoint DW</td>
<td>PCI Dual Channel Ultra &amp; Wide SCSI adapter</td>
</tr>
</tbody>
</table>

CONFIGURATION
The driver attempts to configure itself in accordance with the information found in the configuration file flashpt.conf.

FILES
/kernel/drv/flashpt.conf flashpt device driver configuration file

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO
driver.conf(4), sysbus(4), attributes(5)
NAME  gld – Generic LAN Driver

SYNOPSIS  
#include <sys/stropts.h>

#include <sys/stream.h>

#include <sys/dlpi.h>

#include <sys/ethernet.h>

#include <sys/gld.h>

DESCRIPTION  gld is a multi-threaded, clonable Generic LAN Driver support module which resides in a loadable kernel module, /kernel/misc/gld. It implements most of the STREAMS functions and DLPI functionality required of a LAN driver. Several Solaris network drivers are implemented using GLD.

Any Solaris network driver implemented using GLD is divided into two distinct parts: a generic part that deals with STREAMS and DLPI interfaces, and a device-specific part 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. After the driver has been successfully loaded, it is a fully DLPI-compliant driver.

The device-specific part of the driver calls GLD functions when it receives data or needs some service from GLD. GLD makes indirect calls into the device-specific driver through pointers provided to GLD by the device-specific driver when it registered itself with GLD.

DLPI is an implementation of a standard that defines the services provided by the data link layer. The data link interface is the boundary between the network and the data link layers of the OSI Reference Model. The network layer entity is the user of the services of the data link interface (DLS user). The data link layer entity is the provider of those services (DLS provider). The DLS user accesses the provider using open(9E) to establish a STREAM to the DLS provider. The STREAM acts as a communication channel between a DLS user and a DLS provider.

gld and Ethernet V2 and 802.3  Ethernet V2 service and 802.3 mode are provided by GLD. 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 will communicate with the service provider. SAP values in the range [0-1500] are treated as equivalent and represent a desire by the user for 802.3 mode. If the value of the SAP field of the DL_BIND_REQ is within this range, GLD computes the length, not including initial M_PROTO message blocks, of all subsequent DL_UNITDATA_REQ messages and transmits 802.3 frames having...
this value in the MAC frame header type field. All frames received from the media having a type field in this range are assumed to be 802.3 frames and are routed up all open STREAMS that are bound to any SAP value within this range. 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.

GLD implements both Style 1 and Style 2 providers. The Style 1 provider assigns a physical point of attachment (PPA) based on the major/minor device that has been opened and bound. A 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 2 provider requires the DLS user to explicitly identify the desired PPA using DL_ATTACH_REQ. 

open(9E) creates a STREAM between GLD and the device and DL_ATTACH_REQ, then associates a particular PPA with that STREAM. Style 2 is denoted by a minor number of zero. If the minor number is not zero, it denotes the PPA number plus one. In both Style 1 and Style 2 opens, the device is cloned.

GLD implements a connectionless-mode service. Once the STREAM is initialized, connectionless data transfer can begin. Because there is no established connection, the DLS user must identify the destination of each data unit to be transferred.

GLD implements the following DLPI primitives:

The DL_INFO_REQ primitive requests information about the DLPI STREAM. The message consists of one M_PCPROTO message block. GLD-based drivers return device-dependent values in the DL_INFO_ACK response to this request. However, all GLD-based drivers return the following values:

- The version is DL_VERSION_2.
- The service mode is DL_CLDLS.
- The provider style is DL_STYLE1 or DL_STYLE2.
- No optional Quality Of Service (QOS) support is present, so the QOS fields are zero.

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. This request may not be issued when using the driver in Style 1 mode. Upon completion, the state changes from DL_UNATTACHED to DL_UNBOUND. The message consists of one M_PROTO message block.

The DL_DETACH_REQ primitive requests detach from the physical point of attachment from a STREAM.
The DL_BIND and DL_UNBIND primitives bind and unbind a DLSAP to the STREAM. The PPA associated with each STREAM will be initialized upon completion of the processing of the DL_BIND_REQ. Multiple STREAMS may be bound to the same SAP; each 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. These primitives are accepted by the driver in any state following DL_ATTACHED.

The DL_PROMISCON_REQ and DL_PROMISCOFF_REQ primitives enables and disables 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.

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_PHYS_ADDR_REQ primitive returns the 6-octet Ethernet address currently associated (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 6-octet Ethernet address currently associated (attached) to this STREAM. The credentials of the process which originally opened the STREAM must be superuser or an EPERM error 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 on this STREAM is successful. Once changed, all STREAMS 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 system is rebooted, whichever occurs first.

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 interface between GLD and GLD-based drivers is an internal interface not currently published for external use.

FILES
/kernel/misc/gld loadable kernel module

163 SunOS 5.7 Last modified 1 Jan 1997
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO

`attributes(5), dlpi(7P), attach(9E), open(9E)`

WARNINGS

Contrary to the DLPI specification, GLD returns the device’s correct address length and broadcast address in `DL_INFO_ACK` even before the device has been attached.
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 Symbios 53c875 SCSI chip.

It supports the standard functions provided by the SCSA interface. That is, it supports tagged and untagged queuing, Wide/Fast/Ultra SCSI, 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, and SCSI_OPTIONS_FAST20.

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

A sample case of 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"
```
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:

```bash
# 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:

```bash
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.
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 glm hardware (53C875) supports Wide, Fast, and Ultra SCSI mode. The maximum SCSI bandwidth is 40 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 \texttt{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 <0x00000888>.
    name <cache-line-size> length <4>
      value <0x00000100>.
```
# hdio(7I) - SMD and IPI disk control operations

## Synopsis

```c
#include <sys/hdio.h>
```

## Description

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.

```c
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 (ctrlr specific) */
    uchar_t hdkt_drstat; /* drive status (ctrlr specific, ro) */
};
```

**HDKIOCSTYPE**

The argument is a pointer to a `hdk_type` structure (described below). This `ioctl()` sets specific information about the hard disk.

```c
/*
 * 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 (ctrlr specific) */
    uchar_t hdkt_drstat; /* drive status (ctrlr 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.

```c
struct hdk_badmap {
    caddr_t hdkb_bufaddr; /* address of user's map buffer */
};
```

**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.

```c
/*
 * 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.

```c
/*
 * 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), ipi(7D), xd(7D), xy(7D)
**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.

The cloning character-special device /dev/hme is used to access all SUNW, hme controllers installed within the system.

**APPLICATION PROGRAMMING INTERFACE**

hme and DLPI

The hme driver is a “style 2” Data Link Service provider. All MPROTO 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 (**0xFFFFFFFF**).

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.

The link can be in one of the 4 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 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 and 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

<table>
<thead>
<tr>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/hme</td>
<td>hme special character device.</td>
</tr>
<tr>
<td>/kernel/drv/hme.conf</td>
<td>System wide default device driver properties</td>
</tr>
</tbody>
</table>

### SEE ALSO

ndd(1M), netstat(1M), driver.conf(4), dlpi(7P), ie(7D), le(7D)
NAME
hsfs – High Sierra & ISO 9660 CD-ROM filesystem

DESCRIPTION
HSFS is a filesystem type that allows users access to files on High Sierra or ISO 9660 format CD-ROM disks from within the SunOS operating system. Once mounted, a HSFS filesystem provides standard SunOS read-only file system operations and semantics. That is, users can read files 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 filesystem also contains support for the Rock Ridge Extensions. If the extensions are contained on the CD-ROM, then the filesystem will provide all of the filesystem semantics and file types of UFS, except for writability and hard links.

HSFS filesystems are mounted either with the command:

```
    mount -F hsfs -o ro device-special
directory-name
```

or

```
    mount /hsfs
```

if a line similar to

```
    /dev/dsk/c0t6d0s0 /hsfs hsfs -no ro
```

is in your /etc/vfstab file (and /hsfs exists).

Normally, if Rock Ridge extensions exist on the CD-ROM, the filesystem will automatically use those extensions. If you do not want to use the Rock Ridge extensions, use the “nrr” (No Rock Ridge) mount option. The mount command would then be:

```
    mount -F hsfs -o ro,nrr device-special
directory-name
```

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 nomapcase` option to `mount(1M)`. (See `mount_hsfs(1M)`).

If the CD-ROM contains Rock Ridge 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 will be 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 filesystem; however, despite any write permissions, the file system is read-only, with EROFS returned to any write operations.

High Sierra and ISO 9660 CD-ROMs only support regular files and directories, thus HSFS only supports these file types. 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**

Samply display of filesystem 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 filesystem.

If there are three files

```
BAR.BAZ;1
BAR.BAZ;2
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)`


**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

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 filesystem.
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  
i2o_bs – Block Storage OSM for I2O

SYNOPSIS  
disk@local target id#:a through u

disk@local target id#:a through u raw

DESCRIPTION  
The I2O Block Storage OSM abstraction (BSA, which also is referred to as block storage class) layer is the primary interface that Solaris operating systems use to access block storage devices. A block storage device provides random access to a permanent storage medium. The i2o_bs device driver uses I2O Block Storage class messages to control the block device; and provides the same functionality (ioctl's, for example) that is present in the Solaris device driver like 'cmdk, dadk' on x86 for disk. The maximum size disk supported by i2o_bs is the same as what is available on x86.

The i2o_bs is currently implemented version 1.5 of Intelligent IO specification.

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; the names of the raw files are found in /dev/rdsk.

I2O associates each block storage device with a unique ID called a local target id that is assigned by I2O hardware. This information can be acquired by the block storage OSM through I2O Block Storage class messages. For Block Storage OSM, nodes are created in /devices/pci#/pci#/ which include the local target ID as one component of device name that the node refers to. However the /dev names and the names in /dev/dsk and /dev/rdsk do not encode the local target id in any part of the name.

For example, you might have the following:

```
/devices/pci0,0/pci101e,0@10,1/disk@10:a /dev/dsk/c1d0s0
```

I/O requests to the 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 is used 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.
Fdisk partition 0 is to access the entire disk and is generally used by the fdisk(1M) program.

FILES

/dev/dsk/cndn[s|p]n  block device
/dev/rdsk/cndn[s|p]n  raw device

where:

- **cn**: controller n
- **dn**: instance number
- **sn**: UNIX system slice n (0-15)
- **pn**: fdisk partition (0)

/kernel/drv/i2o_bs  i2o_bs driver
/kernel/drv/i2o_bs.conf  Configuration file

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO

fdisk(1M), format(1M) mount(1M), lseek(2), read(2), write(2), readdir(3C), vfstab(4), acct(4), attributes(5), dkio(7I)
NAME
i2o_scsi – an I2O OS specific module that supports SCSA interface.

DESCRIPTION
The i2o_scsi OSM module is a SCSI HBA driver that supports the SCSA interface. It supports both SCSI Adapter Class and SCSI Peripheral Class functions. It translates the SCSI packet coming down from the SCSA into an I2O SCSI Peripheral Class message, passes it along to the IOP which in turn passes it to the HDM (hardware specific module).

It also uses SCSI Adapter Class functions to manage the SCSI adapter and SCSI bus. For each SCSI Adapter Class I2O device (a SCSI controller), it claims the SCSI Peripheral class devices which are attached to that port. The existing SCSI target drivers which use the SCSA interface should only work with i2o_scsi. This includes target drivers like sd, st, and so on.

FILES
/kernel/drv/i2o_scsi.conf configuration file for the i2o_scsi driver; there are no user-configurable options in this file

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO
attributes(5)
Information Library for Solaris 2.6 (Intel Platform Edition)

NOTES
Throughout the release, support of additional devices may be added. See the Hardware Compatibility List for Solaris 2.6 (Intel Platform Edition) for additional information.
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(3N). 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 protocol 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(3N), recv(3N), send(3N), t_sndudata(3N), t_rcvudata(3N), inet(7P), ip(7P), routing(7P)


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
iee – Intel EtherExpress 16 Ethernet device driver

### SYNOPSIS
```c
#include <sys/stropts.h>
#include <sys/ethernet.h>
#include <sys/dlpi.h>
#include <sys/gld.h>
```

### DESCRIPTION
The iee Ethernet driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface, `dlpi(7P)`, over Intel EtherExpress 16 Ethernet controllers. Multiple EtherLink 16 controllers installed within the system are supported by the driver. The iee driver provides basic support for the EtherLink 16 hardware. Functions include chip initialization, frame transmit and receive, multicast and "promiscuous" support, and error recovery and reporting.

The cloning, character-special device `/dev/iee` is used to access all EtherLink 16 devices installed within the system.

#### iee and DLPI
The iee driver is dependent on `/kernel/misc/gld`, a loadable kernel module that provides the iee 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 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/iee` character special device
- `/kernel/drv/iee.conf` configuration file of iee driver
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO attributes(5), dlpi(7P), gld(7D)
NAME | ieeef – Intel EtherExpress Flash32/82596 Ethernet device driver

SYNOPSIS | 
#include <sys/stropts.h>  
#include <sys/ethernet.h>  
#include <sys/dlpi.h>  
#include <sys/gld.h>

DESCRIPTION | The ieeef Ethernet driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface, dlpi(7P), over Intel EtherExpress Flash32 Ethernet controllers, or the Unisys family of on-motherboard and add-on ethernet implementations using the Intel 82596 network controller.

Multiple controllers installed within the system are supported by the driver. The ieeef driver provides basic support for the above mentioned hardware. Functions include hardware initialization, frame transmit and receive, multicast and "promiscuous" support, and error recovery and reporting.

The cloning, character-special device /dev/ieef is used to access all EtherExpress Flash32/82596 devices installed within the system.

ieef and DLPI | The ieeef driver is dependent on /kernel/misc/gld, a loadable kernel module that provides the ieeef 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 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.

FILES | /dev/ieef | ieeef character special device  
/kernel/drv/ieef.conf | ieeef configuration file
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO  attributes(5), dlpi(7P), gld(7D)
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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SPARC</td>
</tr>
</tbody>
</table>

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 \texttt{dma} handle to the \texttt{cmd} area. Driver did not attach to device; SCSI devices will be inaccessible.

Cannot alloc fcal handle
Driver was unable allocate a \texttt{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 \texttt{fcal} port handle to the memory used for obtaining port database. Driver did not attach to device; SCSI devices will be inaccessible.

\texttt{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 (IP) must be STREAMS devices conforming to the Datalink 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)). This may be initiated by the kernel at boot time or by a user program some time after the system is running. Each interface must be assigned an IP address with the SIOCSIFADDR 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, 10Mb/s 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.

IOCTLS

The following ioctl() calls may be used to manipulate IP network interfaces. Unless specified otherwise, the request takes an ifreq structure as its parameter. This structure has the form:

```c
/* Interface request structure used for socket ioctls. All */
/* interface ioctls must have parameter definitions which */
/* begin with ifr_name. The remainder may be interface specific. */
struct ifreq {
#define IFNAMSIZ 16
  char ifr_name[IFNAMSIZ]; /* if name, for example */
  struct sockaddr ifru_addr;
  struct sockaddr ifru_dstaddr;
  char ifru_oname[IFNAMSIZ]; /* other if name */
  struct sockaddr ifru_broadaddr;
  short ifru_flags;
  int ifru_metric;
  char ifru_data[1]; /* interface dependent data */
  int ifru_enaddr[6];
  int ifru_muxid[2]; /* mux id’s for arp and ip */
  int ifru_index; /* interface index */
} ifr_ifru;
#define ifr_addr ifr_ifru.ifru_addr
#define ifr_dstaddr ifr_ifru.ifru_dstaddr
#define ifr_oname ifr_ifru.ifru_oname
#define ifr_broadaddr ifr_ifru.ifru_broadaddr
#define ifr_flags ifr_ifru.ifru_flags
#define ifr_index ifr_ifru.ifru_index
#define ifr_metric ifr_ifru.ifru_metric
```

195 SunOS 5.7 Last modified 7 Jan 1997
#define ifr_data ifr_ifru.ifru_data /* for use by interface */
#define ifr_enaddr ifr_ifru.ifru_enaddr /* ethernet address */

SIOCSIFADDR  Set interface address. Following the address assignment, the “initialization” routine for the interface is called.

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 initially set to the size of the buffer pointed to by ifc_buf. On return it will contain the length, in bytes, of the configuration list.

SIOCGIFNUM  Get number of interfaces. This request returns an integer which is the number of interface descriptions (struct ifreq) that will be returned by the SIOCGIFCONF ioctl; that is, it gives an indication of how large ifc_len has to be.

SIOCSIFMTU  Set the maximum transmission unit (MTU) size for interface. Place the result of this request in ifru_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. Place the result of this request in ifru_metric field.
The **ifconf** structure has the form:

```c
/*
 * Structure used in SIOCGIFCONF request.
 * Used to retrieve interface configuration
 * for machine (useful for programs which
 * must know all networks accessible).
 */
struct ifconf {
    int ifc_len; /* size of associated buffer */
    union {
        caddr_t ifcu_buf; /* buffer address */
        struct ifreq *ifcu_req; /* array of structures returned */
    } ifc_ifcu;
#define ifc_buf ifc_ifcu.ifcu_buf
#define ifc_req ifc_ifcu.ifcu_req
};
```

**ERRORS**

- **EPERM** The effective user id of the calling process in not superuser.
- **ENXIO** The **ifr_name** member of the **ifreq** structure contains an invalid value.
- **EBADADDR** Wrong address family or malformed address.
- **EBUSY** For **SIOCSIFFLAGS**, this error is returned when the order of bringing the primary/physical interface (for example, le0) up and down is violated. The physical interface must be configured up first.
and cannot be configured down until all the corresponding logical interfaces have been configured down.

**EINVAL**

For `SIOCGIFCONF`, this error is returned when the size of the buffer pointed to by the `ifc_buf` member of the `ifconf` structure is too small.

For `SIOCSIFMTU`, 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.

**SEE ALSO**

`ifconfig(1M)`  `in.routed(1M)`  `ioctl(2)`  `arp(7P)`  `dlpi(7P)`  `ip(7P)`  `streamio(7I)`
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 comprises 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(3N)).

Addresses in the Internet protocol family use the sockaddr_in structure, which has that following members:

```c
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(3N).

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(3N) 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(3N) 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(3N) 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 via.

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(3N) call is restricted to be greater than IPPROTO_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 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 IPPROTO_RESERVED. A unique local port address is also picked when a socket which is not bound is used in a connect(3N) or sendto (see send(3N)) call. This allows programs which do not care which local port number is used to set up TCP connections by simply calling socket(3N) and then connect(3N), and to send UDP datagrams with a socket(3N) 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(3N)).

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(3N), byteorder(3N), connect(3N), gethostbyname(3N), getnetbyname(3N), getprotobyname(3N), getservbyname(3N), getsockopt(3N), send(3N), socket(3N), arp(7P), icmp(7P), ip(7P), tcp(7P), udp(7P)


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
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 via the socket interface (using a “raw socket”) or the Transport Level Interface (TLI). The protocol options defined in the IP specification may be set in outgoing datagrams.

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(3N) and recv(3N)), although the connect(3N) call may also be used to fix the destination for future datagrams (in which case the read(2) or recv(3N) and write(2) or send(3N) 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.

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(3N)). The getsockopt(3N) 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_ADD_MEMBERSHIP**
Join a multicast group.

**IP_DROP_MEMBERSHIP**
Leave a multicast group.

These options take a `struct ip_mreq` as the parameter. The structure contains a multicast address which has to 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 to join on.

**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 will use 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 will use 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. 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 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.

The IP layer will normally act as a router (forwarding datagrams that are not addressed to it, among other things) when the machine has two or more interfaces that are up. This behavior can be overridden by using `ndd(1M)` to set the `/dev/ip` variable, `ip_forwarding`. The value 0 means do not forward; the value 1 means forward. The initialization scripts (see `/etc/init.d/inetinit`) set this value at boot time based on the number of "up" interfaces, but will not turn on IP forwarding at all if the file `/etc/notrouter` exists. When the IP module is loaded, `ip_forwarding` is 0 and remains so if:

- only one non-DHCP-managed interface is up (the most common case)
- the file `/etc/notrouter` exists and DHCP does not say that IP forwarding is on
- the file `/etc/defaultrouter` exists and DHCP does not say IP forwarding is on

Otherwise, `ip_forwarding` will be set to 1.

The IP layer will send 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 will be 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 will be 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 will also be 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.

SEE ALSO

ndd(1M), read(2), write(2), bind(3N), connect(3N),
getsockopt(3N), recv(3N), send(3N), defaultrouter(4), icmp(7P),
if_tcp(7P), inet(7P), routing(7P), tcp(7P), udp(7P)

Braden, R., Requirements for Internet Hosts − Communication Layers, RFC 1122, Information Sciences Institute, University of Southern California, October 1989.


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.

EADDRINUSE A bind() operation was attempted on a socket with a network address/port pair that has already been bound to another socket.

EADDRNOTA VAIL 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 via `connect()`, or to send a datagram via `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.

**ENOBUSFS**

The system ran out of memory for fragmentation buffers or other internal data structures.

**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
iprb – Intel 82557 (D100)-controlled Network Cards

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 controllers. Multiple 82557 controllers
installed within the system are supported by the driver. The iprb driver
provides basic support for the 82557 hardware. Functions include chip
initialization, frame transmit and receive, multicast support, and error recovery
and reporting.

APPLICATION
PROGRAMMING
INTERFACE
iprb and DLPI

The cloning, character-special device /dev/iprb is used to access all 82557
devices installed within the system.

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
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:

- The maximum SDU is 1500 (ETHERMTU).
- The minimum SDU is 0. The 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

CONFIGURATION
The iprb driver does not support the use of shared RAM on the board.

FILES
/dev/iprb
iprb
character special device

/kernel/drv/iprb.conf configuration file of iprb driver
<sys/stropts.h>
<sys/ethernet.h>
<sys/dlpi.h>

207 SunOS 5.7 Last modified 21 May 1997
<sys/gld.h>

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`attributes(5), dlpi(7P), gld(7D)`
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 Hn-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:

```c
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:

```c
typedef enum {
    ISDN_CHAN_NONE = 0x0, /* No channel given */
    ISDN_CHAN_SELF,  /* The channel performing the ioctl */
    ISDN_CHAN_HOST, /* Unix STREAM */
    ISDN_CHAN_CTRL_MGT, /* Controller management */
    ISDN_CHAN_TE_MGT, /* TE channel defines */
    ISDN_CHAN_TE_D, /* Receives activation/deactivation */
    ISDN_CHAN_TE_D_TRACE, /* Trace device for protoanalysis apps */
} 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:

```c
/* 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;
```
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:

```c
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 `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` 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 interface 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.

```c
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**

`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`.

```c
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 */
} isdn_interface_info_t;
```
isdnio(7I) Ioctl Requests

unsigned int mph_di; /* Management: Deactivation Ind */
unsigned int mph_el1; /* Management: Error 1 Indication */
unsigned int mph_el2; /* Management: Error 2 Indication */
unsigned int mph_li_c; /* Management: Info Ind, connection */
unsigned int mph_li_d; /* Management: Info Ind, disconnect */
} isdn_interface_info_t;

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. \textit{fd} is any file descriptor. \textit{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 \textit{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 \textit{fd} is the file descriptor for a management device. \textit{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 \textit{fd} is the file descriptor for a management device. \textit{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 */
    ISDN_PARAM_POWER, /* Take the interface online or offline */
    ISDN_PARAM_PAUSE, /* Paused if == 1, else not paused == 0 */
} isdn_param_tag_t;

217 SunOS 5.7 Last modified 7 Apr 1998
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;

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_T1 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_T2 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_T3 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_T4 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 a file descriptor for an ISDN interface’s management device. arg is a pointer to an isdn_loopback_request_t structure.

```c
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:
ioctl Requests

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:

```c
/* 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 */
```
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.

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.

The following ioctlts describe operations on individual channels and the connection of multiple channels.

### ISDN_SET_FORMAT

It 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`.

```c
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 intended operation */
- EINVAL /* Invalid format request */
- EIO /* Set format attempt failed. */

The ISDN_SET_CHANNEL ioctl sets up a data connection within an ISDN controller. The ISDN_SET_CHANNEL ioctl can only be issued from 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.

```c
/* Number of directions for data flow */
typedef enum {
    ISDN_PATH_NOCHANGE = 0, /* Invalid value */
    ISDN_PATH_DISCONNECT, /* Disconnect data path */
    ISDN_PATH_ONEDWAY,  /* 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 intended operation */
- EBUSY /* Connection in use */
- EINVAL /* Invalid connection request */
- EIO /* Connection attempt failed. */

The ISDN_IOCTL_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_IOCTL_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.

The ISDN_IOCTL_GETCONFIG ioctl is used to get the current connection status of all ISDN channels associated with a particular management STREAM. ISDN_IOCTL_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:

```c
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), dbri(7D), streamio(7I)

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.

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
A sample program.

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# 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
```

The above example is more specific to the ISP controller on SBus. To achieve the same setting of scsi-options 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;
```

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/c4t3d0s0
lrwxrwxrwx 1 root root 58 Jun 20 23:48 /dev/rdsk/c4t3d0s0 ->
../../devices/pci@1f,2000/pci@1/SUNW,isptwo@4/sd@3,0:a,raw
```

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";
```

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.

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` ELK Kernel Module

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

Attributes

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>SPARC</td>
</tr>
</tbody>
</table>

See Also

`prtconf(1M)`, `driver.conf(4)`, `attributes(5)`

`scsi_abort(9F)`, `scsi_hba_attach(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

OpenBoot Command Reference

227

SunOS 5.7

Last modified 12 Jan 1998
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 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. This 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.
The \texttt{isp} driver exports properties indicating per target the negotiated transfer speed (\texttt{target<n>-sync-speed}), whether tagged queuing has been enabled (\texttt{target<n>-TQ}), and whether the wide data transfer has been negotiated (\texttt{target<n>-wide}). The \texttt{sync-speed} property value is the data transfer rate in KB/sec. The \texttt{target-TQ} and \texttt{target-wide} properties have no value. The existence of these properties indicate that tagged queuing or wide transfer has been enabled. Refer to \texttt{prtconf}(1M) (verbose option) for viewing the \texttt{isp} properties.

\begin{verbatim}
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>.
\end{verbatim}
# NAME
kb – keyboard STREAMS module

## SYNOPSIS

```c
#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>
```

## DESCRIPTION

The `kb` STREAMS module processes byte streams generated by keyboard attached to a CPU serial port. Definitions for altering keyboard translation, and reading events from the keyboard, are in `<sys/kbio.h>` and `<sys/kbd.h>`.

`kb` recognizes which keys have been typed using a set of tables for each known type of keyboard. Each translation table is an array of 128 16-bit words (unsigned shorts). If an entry in the table is less than 0x100, it 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 used to convert raw keystation codes to event values. The tables are:

- **Unshifted**
  - Used when a key is depressed and no shifts are in effect.

---

Last modified 10 Mar 1998

SunOS 5.7

232
Shifted  Used when a key is depressed and a Shift key is being 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 being 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 being 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 which is a number from 0 to 127. This number is used as an index into the translation table that is currently in effect. If the corresponding entry in that translation table is a value from 0 to 255, this value is treated as an ISO 8859/1 character, and that character is the result of the translation.

If the entry is a value above 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 in the list below. The value of the low-order bits, when added to this constant, distinguishes between keys within each class:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHIFTKEYS</td>
<td>A shift key. The value of the particular shift key is added to determine which shift mask to apply:</td>
</tr>
<tr>
<td>CAPSLOCK</td>
<td>&quot;Caps Lock&quot; key.</td>
</tr>
<tr>
<td>SHIFTLOCK</td>
<td>&quot;Shift Lock&quot; key.</td>
</tr>
<tr>
<td>LEFTSHIFT</td>
<td>Left-hand &quot;Shift&quot; key.</td>
</tr>
<tr>
<td>RIGHTSHIFT</td>
<td>Right-hand &quot;Shift&quot; key.</td>
</tr>
<tr>
<td>LEFTCTRL</td>
<td>Left-hand (or only) &quot;Control&quot; key.</td>
</tr>
<tr>
<td>RIGHTCTRL</td>
<td>Right-hand &quot;Control&quot; key.</td>
</tr>
<tr>
<td>ALTGRAPH</td>
<td>&quot;Alt Graph&quot; key.</td>
</tr>
</tbody>
</table>

233 SunOS 5.7 Last modified 10 Mar 1998
STREAMS Modules

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
This key is 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.
FA_CLASS 0x400
This key is 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:

KTABLE_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
Function keys. The next-to-lowest 4 bits indicate the group of function keys:
**LEFTFUNC 0x600**  
**RIGHTFUNC 0x610**  
**TOPFUNC 0x620**  
**BOTTOMFUNC 0x630**

The low 4 bits indicate the function key number within the group:
- LF\(n\)  
  \((\text{LEFTFUNC}+\(n\)-1))
- RF\(n\)  
  \((\text{RIGHTFUNC}+\(n\)-1))
- TF\(n\)  
  \((\text{TOPFUNC}+\(n\)-1))
- BF\(n\)  
  \((\text{BOTTOMFUNC}+\(n\)-1))

There are 64 keys reserved for function keys. The actual positions may not be on left/right/top/bottom of the keyboard, although they usually are.

**PADKEYS 0x700**

This key is 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 as follows:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x700</td>
<td>PADEQUAL &quot;=&quot; key</td>
</tr>
<tr>
<td>0x701</td>
<td>PADSLASH &quot;/&quot; key</td>
</tr>
<tr>
<td>0x702</td>
<td>PADSTAR &quot;*&quot; key</td>
</tr>
<tr>
<td>0x703</td>
<td>PADMINUS &quot;,&quot; key</td>
</tr>
<tr>
<td>0x704</td>
<td>PADSEP &quot;,,&quot; key</td>
</tr>
<tr>
<td>0x705</td>
<td>PAD7 &quot;7&quot; key</td>
</tr>
<tr>
<td>0x706</td>
<td>PAD8 &quot;8&quot; key</td>
</tr>
<tr>
<td>0x707</td>
<td>PAD9 &quot;9&quot; key</td>
</tr>
<tr>
<td>0x708</td>
<td>PADPLUS &quot;+&quot; key</td>
</tr>
<tr>
<td>0x709</td>
<td>PAD4 &quot;4&quot; key</td>
</tr>
<tr>
<td>0x70A</td>
<td>PAD5 &quot;5&quot; key</td>
</tr>
<tr>
<td>0x70B</td>
<td>PAD6 &quot;6&quot; key</td>
</tr>
<tr>
<td>0x70C</td>
<td>PAD1 &quot;1&quot; key</td>
</tr>
<tr>
<td>0x70D</td>
<td>PAD2 &quot;2&quot; key</td>
</tr>
<tr>
<td>0x70E</td>
<td>PAD3 &quot;3&quot; key</td>
</tr>
</tbody>
</table>
In TR_ASCII mode, when a function key is pressed, 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

kb is in “compatibility mode” when it starts up. In this 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 8th bit set) are presented as events with codes in the ISO_FIRST range (as defined in `<sys/vuid_event.h>`). The event code is ISO_FIRST plus the character value. This is for backwards compatibility with older versions of the keyboard driver. If compatibility mode is turned off, ISO 8859/1 characters are presented as events with codes equal to the character code.

### IOCTLS

The following ioctl() requests set and retrieve the current translation mode of a keyboard:

- **KIOCTRANS**: The argument is a pointer to an int. The translation mode is set to the value in the int pointed to by the argument.
- **KIOCGRTRANS**: The argument is a 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:

```c
struct kiockeymap {
    int kio_tablemask; /* Translation table (one of: 0, CAPSMASK,
                        * SHIFTMASK, CTXLMASK, UPMASK,
                        * ALTGRAPHMASK, NUMLOCKMASK)
    
    #define KIOCABORT1 --1 /* Special ‘mask’: abort1 keystation */
}
```

Last modified 10 Mar 1998
#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 The argument is a pointer to a kiocemap structure. The
translation table entry referred to by the values in that
structure is changed.

kio_tablemask specifies which of the five
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)

kio_station 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.

There are a couple special values of
kio_tablemask that 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

239 SunOS 5.7 Last modified 10 Mar 1998
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.

KIOCGK
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.

KIOCTY
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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KB_SUN3</td>
<td>Sun Type 3 keyboard</td>
</tr>
<tr>
<td>KB_SUN4</td>
<td>Sun Type 4 keyboard</td>
</tr>
<tr>
<td>KB_ASCII</td>
<td>ASCII terminal masquerading as keyboard</td>
</tr>
<tr>
<td>KB_PC</td>
<td>Type 101 PC keyboard</td>
</tr>
<tr>
<td>KB_DEFAULT</td>
<td>KB_DEFAULT is 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.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KBD_CMD_RESET</td>
<td>Reset keyboard as if power-up.</td>
</tr>
<tr>
<td>KBD_CMD_BELL</td>
<td>Turn on the bell.</td>
</tr>
<tr>
<td>KBD_CMD_NOBELL</td>
<td>Turn off the bell.</td>
</tr>
</tbody>
</table>
STREAMS Modules

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 we cannot query the keyboard, and also because a process could do writes to the appropriate serial driver — thus going around this ioctl() request) we do not provide an equivalent ioctl() to query its state.

KIOCSLED
The argument is a pointer to a 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 of the Japanese layouts, the value for the fifth LED is:

LED_KANA
“Kana” light.

KIOCGL
The argument is a pointer to a char. The current state of the LEDs is stored in the char pointed to by the argument.

KIOCSCOMPAT
The argument is a 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.
The argument is a pointer to an \texttt{int}. The current state of "compatibility mode" is stored in the \texttt{int} pointed to by the argument. The following \texttt{ioctl()} request allows the default effect of the keyboard abort sequence to be changed.

\texttt{KIOCSKABORTEN}  

The argument is a pointer to an \texttt{int}. The keyboard abort sequence (typically \texttt{L1-A} or \texttt{Stop-A} on the keyboard on SPARC systems and \texttt{BREAK} on the serial console device) effect is enabled if the \texttt{int} has a non-zero value, otherwise, the keyboard abort sequence effect is disabled. When enabled, the default effect 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. The default effect may be different on server systems with key switches when the key switch is in the 'secure' position. On these server systems, the effect is always 'disabled' when the key switch is in the 'secure' position. This ioctl returns \texttt{EPERM} if the caller is not the superuser.

These \texttt{ioctl()} requests are supported for compatibility with the system keyboard device /dev/kbd.

\texttt{KIOCSDIRECT}  

Has no effect.

\texttt{KIOCGDIRECT}  

Always returns 1.

\textbf{ATTRIBUTES}  

See \texttt{attributes(5)} for descriptions of the following attributes:

\begin{tabular}{|c|c|}
\hline
\textbf{ATTRIBUTE TYPE} & \textbf{ATTRIBUTE VALUE} \\
\hline
Architecture & SPARC \\
\hline
\end{tabular}

\textbf{SEE ALSO}  

\texttt{kbd(1), loadkeys(1), kadb(1M), keytables(4), attributes(5), termio(7I)}

\textbf{NOTES}  

Many of the keyboards released after Sun Type 4 keyboard also report themselves as Sun Type 4 keyboard.

Last modified 10 Mar 1998  

SunOS 5.7  

242
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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO     attributes(5), vuidmice(7M)
NAME  keyboard – system console keyboard

DESCRIPTION  keyboard is a component of the kd driver, which is comprised of the display and keyboard drivers.

The Solaris for x86 keyboard may be either an 84- or a 101-key standard PC keyboard. When the system is booting, keyboard services are provided by the keyboard section of the kd driver.

Developers are not encouraged to write programs that communicate directly with the keyboard; they should make use of the environment provided by the windows server.

FILES  /dev/console

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO  attributes(5), console(7D), display(7D)
NAME        kstat – kernel statistics driver

DESCRIPTION The kstat driver is the mechanism used by the kstat(3K) library to extract
kernel statistics. This is NOT a public interface.

FILES       /dev/kstat     kernel statistics driver

SEE ALSO    kstat(3K), 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(3E) 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(3E), kvm_open(3K), 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(3K)` 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 ksym's 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 ksym's driver to be loaded. In this case, the file size will appear to be zero. A solution for this behavior is to first open the /dev/ksyms file, causing the ksym's 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(3K)` for details.
ldterm – standard STREAMS terminal line discipline module

#include <sys/stream.h>
#include <sys/termios.h>

DESCRIPTION
ldterm is a STREAMS module that provides most of the termio(7I) terminal interface. This 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. ldterm performs all other termio/termios functions; some of them, however, 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.

ldterm also handles Extended Unix Code (EUC) and multi-byte characters.

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

When this message is received, 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.

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
ldterm(7M) STREAMS Modules

the read queue. If the signal is also to flush output, an M_FLUSH message with a flag byte of FLUSHW is sent downstream.

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 then on will process only those flags which 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 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 ioctl 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.
All other messages are passed downstream unchanged.

IOCTLS
ldterm 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.

TCFLUSH

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 \texttt{M\_STOP} message is sent downstream. If the argument is 3 and input is stopped, an \texttt{M\_START} message is sent downstream.

**TCSBRK**

The message is passed downstream, so the driver has a chance to drain the data and then send an \texttt{M\_IOCACK} message upstream.

**EUC\_WSET**

This call takes a pointer to an \texttt{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 \texttt{M\_CTL} messages.

**EUC\_WGET**

This call takes a pointer to an \texttt{eucioc} structure, and returns in it the EUC code set widths currently in use by the EUC line discipline.

**SEE ALSO**

\texttt{termios(3), console(7D), termio(7I)}

*STREAMS Programming Guide*
**NAME**
le, lebuffer, ledma – Am7990 (LANCE) Ethernet device driver

**SYNOPSIS**
/dev/le

**DESCRIPTION**
The Am7990 (LANCE) Ethernet driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface, dlpi(7P), over a LANCE Ethernet controller. The motherboard and add-in SBus LANCE controllers of several varieties are supported. Multiple LANCE controllers installed within the system are supported by the driver. The le driver provides basic support for the LANCE hardware. Functions include chip initialization, frame transmit and receive, multicast and promiscuous support, and error recovery and reporting.

**APPLICATION PROGRAMMING INTERFACE**
The cloning character-special device /dev/le is used to access all LANCE controllers installed within the system.

The lebuffer and ledma device drivers are bus nexus drivers which cooperate with the le leaf driver in supporting the LANCE hardware functions over several distinct slave-only and DVMA LANCE-based Ethernet controllers. The lebuffer and ledma bus nexi drivers are not directly accessible to the user.

**le and DLPI**
The le 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) on 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 value is −2 meaning the physical address component is followed immediately by a 2 byte sap component within the DLSAP address.

Last modified 23 Aug 1994 SunOS 5.7 252
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 (0xFFFFFFFF).

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 le 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 le 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 le driver. The le 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.

**le 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.

**FILES**

/dev/le

le special character device.

/kernel/drv/options.conf

System wide default device driver properties

**SEE ALSO** netstat(1M), driver.conf(4), dlpi(7P), ie(7D)

Last modified 23 Aug 1994
SPARCstation 10 Twisted-Pair Ethernet Link Test

Twisted-Pair Ethernet Link Test

DIAGNOSTICS

le%d: msg too big: %d

The message length exceeded ETHERMAX.

le%d: Babble error − sent a packet longer than 1518 bytes

While transmitting a packet, the LANCE chip has noticed that the packet’s length exceeds the maximum allowed for Ethernet. This error indicates a kernel bug.

le%d: No carrier − transceiver cable problem?

The LANCE chip has lost input to its carrier detect pin while trying to transmit a packet.

le%d: Memory Error!

The LANCE chip timed out while trying to acquire the bus for a DVMA transfer.

NOTES

If you are using twisted pair Ethernet (TPE), you need to be aware of the link test feature. The IEEE 10Base-T specification states that the link test should always be enabled at the host and the hub. Complications may arise because:

1. Some older hubs do not provide link pulses
2. Some hubs are configured to not send link pulses

Under either of these two conditions the host translates the lack of link pulses into a link failure unless it is programmed to ignore link pulses. To program your system to ignore link pulses (also known as disabling the link test) do the following at the OpenBoot PROM prompt:

```bash
<#0> ok setenv tpe-link-test? false
tpe-link-test? = false
```

The above command will work for SPARCstation-10, SPARCstation-20 and SPARCclassic systems that come with built in twisted pair Ethernet ports. For other systems and for add-on boards with twisted pair Ethernet refer to the documentation that came with the system or board for information on disabling the link test.
SPARCstation-10, SPARCstation-20 and SPARCclassic systems come with a choice of built in AUI (using an adapter cable) and TPE ports. In Solaris 2.2 an auto-selection scheme was implemented in the `le` driver that will switch between AUI and TPE depending on which interface is active. Auto-selection uses the presence or absence of the link test on the TPE interface as one indication of whether that interface is active. In the special case where you wish to use TPE with the link-test disabled you should manually override auto-selection so that the system will use only the twisted pair port.

This override can be performed by defining the `cable-selection` property in the `options.conf` file to force the system to use TPE or AUI as appropriate. The example below sets the cable selection to TPE.

```
example# cd /kernel/drv
example# echo 'cable-selection="tpe";' >> options.conf
```

Note that the standard `options.conf` file contains important information; the only change to the file should be the addition of the `cable-selection` property. Be careful to type this line exactly as shown above, ensuring that you append to the existing file, and include the terminating semi-colon. Alternatively you can use a text editor to append the line

```
cable-selection="tpe";
```

to the end of the file.

Please refer to the `SPARCstation 10 Twisted-Pair Ethernet Link Test (801-2481-10)`, `Twisted-Pair Ethernet Link Test (801-6184-10)` and the `driver.conf(4)` man page for details of the syntax of driver configuration files.
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 dlsap address length is 7.
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.

No optional quality of service (QOS) support is included at present, so
the QOS fields should be initialized to 0.

The provider style is DL_STYLE2.

The DLPI version is DL_VERSION_2.

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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO attributes(5), dlpi(7P)
### NAME
lockstat – kernel lock statistics driver

### DESCRIPTION
The `lockstat` driver is the mechanism used by the `lockstat(1M)` command to extract kernel lock statistics. This is not a public interface.

### FILES
/dev/lockstat kernel lock statistics driver

### SEE ALSO
`lockstat(1M)`

Last modified 7 May 1997
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).

SEE ALSO
mount(1M), chroot(2), mount(2), sysfs(2), vfstab(4)

WARNINGS
Loopback mounts must be used with care; the potential for confusing users and applications is enormous. 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.
BUGS

(1) Files can be modified on a read-only loopback mounted file system and (2) a loopback mounted file system can be unmounted even if there is an open regular file on that file system. The loopback file system works by shadowing directories of the underlying file system. Because no other file types are shadowed, the loopback file system can not enforce read-only access to non-directory files located on a read-only mounted loopback file system. Thus, write access to regular files located on a loopback mounted file system is determined by the underlying file system. In addition, the loopback file system can not correctly determine whether a loopback mounted file system can be unmounted or not. It can only detect when a directory is active or not, not when a file within a directory is active. Thus, a loopback mounted file system may be unmounted if there are no active directories on the file system, even if there are open files on the file system.
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 ar
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(3S) 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;
};

263 SunOS 5.7 Last modified 11 Mar 1998
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
log(7D) Devices

A process may also send a message of the same structure to \texttt{log}, even if it is not an error or trace logger. The only fields of the \texttt{log\_ctl} structure in the control part of the message that are accepted are the \texttt{level}, \texttt{flags}, and \texttt{pri} fields; all other fields are filled in by \texttt{log} before being forwarded to the appropriate logger. The data portion must contain a null terminated format string, and any arguments (up to \texttt{NLOGARGS}) must be packed, 32-bits each, on the next 32-bit boundary following the end of the format string.

\texttt{ENXIO} is returned for \texttt{I\_TRCLOG} \texttt{ioctl}s without any \texttt{trace\_ids} structures, or for any unrecognized \texttt{ioctl} calls. The driver silently ignores incorrectly formatted \texttt{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 \texttt{/dev/conslog}, using either \texttt{write(2)} or \texttt{putmsg(2)}.

**EXAMPLES**

**EXAMPLE 1** \texttt{I\_ERRLOG} registration.

Example of \texttt{I\_ERRLOG} registration:

```c
struct strioctl ioc;
ioc.ic_cmd = I\_ERRLOG;
ioc.ic_timeout = 0; /* default timeout (15 secs.) */
ioc.ic_len = 0;
ioc.ic_dp = NULL;
ioctl(log, I\_STR, &ioc);
```

**EXAMPLE 2** \texttt{I\_TRCLOG} registration.

Example of \texttt{I\_TRCLOG} registration:

```c
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 */
ioctl(log, I\_STR, &ioc);
ioc.ic_cmd = I\_TRCLOG;
ioc.ic_timeout = 0;
ioc.ic_len = 2 \times \text{sizeof(struct trace\_ids)};
ioc.ic_dp = (\text{char} *)tid;
ioctl(log, I\_STR, &ioc);
```

Example of submitting a \texttt{log} message (no arguments):

```c
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 the log driver
/dev/conslog the write only instance of the log driver, for console logging

SEE ALSO strace(1M), strerror(1M), intro(2), getmsg(2), ioctl(2), putmsg(2), write(2), printf(3S), strlog(9F)

STREAMS Programming Guide
NAME
logi – LOGITECH Bus Mouse device interface

SYNOPSIS
/dev/logi

DESCRIPTION
The logi driver supports the LOGITECH Bus Mouse. It allows applications to obtain information about the mouse’s movements and the status of its buttons. The data is read in the Five Byte Packed Binary Format, also called MSC format.

FILES
/dev/logi

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO
attributes(5)
NAME
lp – driver for parallel port

SYNOPSIS
include <sys/bpp_io.h>

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>:

```c
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.

Last modified 21 May 1997 SunOS 5.7 268
FILES
/platform/i86pc/kernel/drv/lp.conf configuration file for lp driver

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO
sysbus(4), attributes(5), streamio(7I)

NOTES
A read operation on a bi-directional parallel port is not supported.
NAME       Item – ANSI Layered Console Driver

SYNOPSIS   #include <sys/types.h>

            #include <fcntl.h>

            #include <visual.h>

            #include <sys/ltem.h>

DESCRIPTION  The ltem driver provides a general-purpose ANSI interface to the system
              console device. ltem is a layered device driver which on one side provides the
              kernel with a consistent interface to the system console device (and therefor to
              the console framebuffer) and on the other side uses ioctl to send data to the
              framebuffer driver (see visual_io(7I)).

IOCTLS      The following ioctl(2) calls are supported:

            VIS_CONS_MODE_CHANGE  Notifies ltem that the resolution of the
                                  underlying framebuffer has been changed. ltem
                                  will stop console output, notify the framebuffer
                                  (by passing this ioctl on), reset the terminal
                                  emulator (using the VIS_DEVFINI and
                                  VIS_DEVINIT ioctls), and allow console output
                                  again.

FILES       /dev/ltem/*  ANSI console layered driver

SEE ALSO    ioctl(2), visual_io(7I)
NAME
m64 – 8-bit PCI color memory frame buffer

SYNOPSIS
SUNW, m64B0@pci-slot:m64X

DESCRIPTION
m64 is the PGX 8-bit color frame buffer and graphics accelerator, with 8-bit colormap. It provides the standard frame buffer interface defined in fbio(7I).

The m64 has registers and memory that may be mapped with mmap(2).

There is extra on-board memory which may be used for scratch-pad, double-buffering or off-screen rendering. The total amount of memory on the board may be found with the FBIOGATTR ioctl. Total mappable memory, including on-screen memory, is attr.sattr.dev_specific[0].

The chip revision number is returned in dev_specific[2].

The dac revision number is returned in dev_specific[3].

The prom revision number is returned in dev_specific[4].

The byte offset from the start of the frame buffer to the start of the visible part of the frame buffer is returned in dev_specific[5].

The m64 frame buffer has a 2-color cursor. The color is determined by the mask and data planes, as written by the FBIOSETCURS ioctl. mask:data combinations are as follows: 0x=transparent, 10=color0, 11=color1.

Maximum cursor size is 64x64 pixels. The Mask and Image pointers in the fbcursor structure should point to data which is zero-padded to 32-bits per scanline and aligned on a 32-bit boundary.

IOCTLS
The m64 frame buffer accepts the following ioctl(2) calls, which are defined in <sys/fbio.h> and <sys/visual_io.h>. All are implemented as described in fbio(7I):

| FBIOGATTR | FBIOGETCMAP |
| FBIOGTYPE | FBIOGGETCMAP |
| FBIOSATTR | FBIOSSVIDEO |
| FBIOGVIDEO | FBIOVERTICAL |
| FBIOSCURSOR | FBIOGSCURSOR |
| FBIOSCURPOS | FBIOGSCURPOS |
| FBIOGCMAX | FBIOGXINFO |
| FBIOMONINFO | FBIOVRTOFFSET |

VIS_GETIDENTIFIER
The value returned by VIS_GETIDENTIFIER is SUNWm64.

FBIOPUTCMAP returns immediately, although the actual colormap update may be delayed until the next vertical retrace. If vertical retrace is currently in progress, the new colormap takes effect immediately.

FBIOPUTCMAP returns immediately with the currently-loaded colormap, unless a colormap write is pending (see above), in which case it waits until the colormap is updated before returning. This may be used to synchronize software with colormap updates.

The size and linebytes values returned by FBIOGATTR, FBIOGTYPE, and FBIOGXINFO are measured in bytes. The proper ways to compute the size of a frame buffer mapping is either:

\[ \text{size}=\text{linebytes}\times\text{height} \]

or to use the size attribute in FBIOGATTR, FBIOGTYPE.

ioctl functions which nominally wait for vertical retrace (FBIOVERTICAL, FBIOPUTCMAP) do not wait, but return immediately, if video is blanked or vertical retrace is not being generated. The vertical retrace counter page is not updated if vertical retrace is not being generated. Vertical retrace is not generated when the device is in energy-saving mode.

FILES
/dev/fbs/m64n A device special file.
/dev/fb The default frame buffer.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>UltraSPARC with a PCI I/O Bus</td>
</tr>
</tbody>
</table>

SEE ALSO ioctl(2), mmap(2), attributes(5), fbio(7I)

Last modified 18 Jun 1997
mem(7D) Devices

NAME
mem, kmem – physical or virtual memory

SYNOPSIS
/dev/mem /dev/kmem

DESCRIPTION
The file /dev/mem is a special file that is an image of the physical memory of the computer. The file /dev/kmem is a special file that is an image of the kernel virtual memory of the computer. Either may be used, for example, to examine, and even patch the system.

Byte addresses in /dev/mem are interpreted as physical memory addresses. Byte addresses in /dev/kmem are interpreted as kernel virtual memory addresses. References to non-existent locations cause errors to be returned (see ERRORS below).

The file /dev/kmem accesses up to 4GB of kernel virtual memory. The file /dev/mem accesses physical memory; the size of the file is equal to the amount of physical memory in the computer. This can be larger than 4GB; in which case, memory beyond 4GB can be accessed using a series of read(2) and write(2) commands or a combination of llseek(2) and read(2) and write(2).

ERRORS
EFAULT Bad address. This error can occur when trying to: write(2) a read-only location, read(2) a write-only location, or read(2) or write(2) a non-existent or unimplemented location.

ENXIO This error results from attempting to mmap(2) a non-existent physical (mem) or virtual (kmem) memory address.

FILES
/dev/mem File containing image of physical memory of computer.
/dev/kmem File containing image of kernel virtual memory of computer.

SEE ALSO
llseek(2), mmap(2), read(2), write(2)

NOTES
Some of /dev/kmem cannot be read because of write-only addresses or unequipped memory addresses.
NAME
mhd – multihost disk control operations

SYNOPSIS
#include <sys/mhd.h>

DESCRIPTION
These ioctl(2) 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.

A non-shared multihost disk can be in one of two states:
1) Exclusive access state, where only one connected host has I/O access, or
2) 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 is running as the "local host," and views all other machines as "remote hosts." For each I/O or ioctl request, the requesting host is the local host.

Each request requires root privilege.

Note that these ioctls were originally designed to work with SCSI disks.

IOCTLs

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, through the use of 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.

MHIOCENFAILFAST  Enables or disables the failfast option in the multihost disk driver and enables or disables automatic probing of a multihost disk, as 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.

**Automatic Probe Function**

The MHIOCENFAILFAST ioctl sets up a timeout in the driver to continuously
schedule automatic probes of a disk. The automatic probe function works by
scheduling the driver to execute the MHIOCSTATUS operation every \( n \)
milliseconds, rounded up to the next integral multiple of the system clock’s
resolution.

If access rights to the multihost disk were expected to be held by the local
host, but the local host no longer has access rights, then the driver
immediately panics the machine in order to comply with the failfast model.

Implementation Note: 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.

**Failfast**

A technique generally used in fault-tolerant systems to diagnose a
malfunctioning component is to externally detect a deviation from that
component’s expected behavior. The recovery model is fault containment
through failfast: the halting, isolation, and removal from service of the bad
component before its potentially incorrect operation can spread the damage
any further.

**RETURN VALUES**

Each request returns −1 on failure and sets errno to indicate the error.

- **EACCES** Access rights were denied.
- **EIO** The multihost disk or controller was unable to successfully
  complete the requested operation.

**USAGE**

```c
#include <sys/mhd.h>
ioctl(fd, MHIOCTKOWN, (struct mhioctkown *)tkown);
ioctl(fd, MHIOCRELEASE);
ioctl(fd, MHIOCSTATUS);
ioctl(fd, MHIOCENFAILFAST, (unsigned int)millisecs);
```

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWhea</td>
</tr>
</tbody>
</table>
SEE ALSO  ioctl(2), attributes(5)
NAME
mic – Multi-interface Chip driver

SYNOPSIS
SUNW,mic@slot,offset:a SUNW,mic@slot,offset:b SUNW,mic@slot,offset:ir

DESCRIPTION
The Multi-interface Chip (MIC) provides two asynchronous serial input/output
channels. These channels provide high speed buffered serial I/O, with optional
hardware flow control support. Baud rates from 110 to 115200 are supported.

The first channel can either be routed through an infra-red port or the a serial
port. If the device is opened using the ir device, the driver routes the first
channel through the infra-red port. If the device is opened using the a device,
the first channel is routed through the a serial port. You cannot use both the a
port and the ir port simultaneously. The second channel (the b serial port) has
no infra-red capability and may be used independently of the first channel.

APPLICATION
PROGRAMMING
INTERFACE
The mic module is a loadable STREAMS driver that provides basic support for
the MIC hardware, together with basic asynchronous communication support.
The driver supports those termio(7I) device control functions specified by
flags in the c_cflag word of the termios structure, excluding HUPCL,
CLOCAL, CIBAUD, CRTSCTS, and PAREXT. The driver does not support device
control functions specified by flags in the c_iflag word of the termios
structure. Specifically, the driver assumes that IGNBRK and IGNPAR are always
set. 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 infra-red port provides access to two different modes of modulation. The
default mode is called pulse mode and is compatible with the Infra-red Data
Association (IrDA) modulation and the Hewlett-Packard Serial Infra-red (SIR)
modulation. The second modulation is called high frequency mode and is
compatible with the Sharp Amplitude Shift Keying (ASK) modulation. The
default modulation when using high frequency mode is 500 KHz.

The character-special devices /dev/term/mic/a and /dev/term/mic/b are
used to access the two serial ports on the MIC chip.

The character-special device /dev/term/mic/ir is used to access the
infra-red port of the chip.

IOCTLS
The standard set of termio ioctl() calls are supported by the mic driver.

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 speeds supported
by termio. The speeds cannot be set independently; when the output speed is
set, the input speed is set to the same speed. To support higher speeds than
defined in termio, the two lowest speeds, B50 and B75, have been remapped to 96000 and 115200 baud respectively.

The following ioctl(2) requests are defined in <sys/micio.h>.

MIOCSLPBK Set SCC loopback mode. This internally loops back transmitted messages within the channel.

MIOCCLPBK Clear SCC loopback mode.

There are six ioctl() calls which are specific to the infra-red port and can only be used when the device has been opened in infra-red mode:

MIOCGETM_IR Returns the current IR mode defined in <sys/micio.h>.

MIOCSETM_IR Takes an additional argument of the desired IR mode (defined in <sys/micio.h>) and sets the port to this mode.

MIOCGETD_IR Returns the current IR carrier divisor. The carrier frequency can be calculated from the divisor and the formula:

\[
\text{carrier frequency} = \frac{19660}{4 \times (\text{divisor} + 1)} \text{ KHz}
\]

MIOCSETD_IR Sets the current IR carrier divisor. The desired frequency can be set by using a divisor calculated by the following formula, where the frequency is specified in KHz:

\[
\text{divisor} = \frac{19660}{\text{frequency} / 4 - 1}
\]

MIOCSLPBK_IR Set IR loopback mode. This enables the receiver during transmit, so that sent messages are also received through the ir port.

MIOCCLPBK_IR Clears IR loopback mode.

ERRORS

The open() function will fail if:

ENXIO The unit being opened does not exist.

EBUSY The channel is in use by another serial protocol. Remember that both the a and ir ports use the same channel.

FILES

/dev/term/mic/a asynonynchronous serial line using port a
/dev/term/mic/b asynchronous serial line using port b

Last modified 1 Jan 1997
/dev/term/mic/ir  asynchronous serial infra-red line using the infra-red port

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>SUN4m, SPARCstation Voyager</td>
</tr>
</tbody>
</table>

SEE ALSO
tip(1), ports(1M), ioctl(2), open(2), attributes(5), ldtterm(7M), termio(7I), ttcompat(7M)

DIAGNOSTICS
mic: Rx FIFO overflow
The driver’s internal 64-character buffer overflowed before it could be serviced.

mic: Rx buffer full - draining FIFO
The driver’s character input buffer overflowed before it could be serviced.

NOTES
Currently hardware flow control is not implemented. The state of DCD, CTS, RTS, and DTR interface signals cannot be queried, nor can hardware flow control be enabled using the CRTSCTS flag in the c_cflag word of the termios structure.
## NAME
mlx – low-level module for Mylex DAC960E EISA and Mylex DAC960P/PD/PD-Ultra/PL PCI host bus adapter series

## SYNOPSIS
/kernel/drv/mlx

## DESCRIPTION
The mlx module provides low-level interface routines between the common disk/tape I/O subsystem and the Mylex DAC960E, and DAC960P/PD/PD-Ultra/PL controllers. The mlx module can be configured for disk, CD-ROM, and streaming tape support for one or more host adapter boards.

## CONFIGURATION
Auto-configuration code determines whether the adapter is present at the configured address and what types of devices are attached to it. The Mylex DAC960E and DAC960P/PD/PD-Ultra/PL are primarily used as disk array (system drive) controllers. In order to configure the attached disk arrays, the controller must first be configured prior to Solaris boot using the configuration utilities provided by the hardware manufacturer. With these utilities, the user can set different levels of redundant arrays of independent disks (RAID), striped parameters, caching mechanisms, and so on. For more information, refer to the user’s manual supplied with your hardware.

Configuration Tips
- The Mylex DAC960E and DAC960P/PD/PD-Ultra/PL BIOS can handle multiple cards. Therefore, if more than one Mylex DAC960Ea or DAC960P/PD/PD-Ultra/PL adapter is installed in a system, only the BIOS of the one in the lowest slot should be enabled and the BIOS in any other adapter should be disabled.
- Enable tag queueing only for the SCSI disk drives that are officially tested and approved by Mylex Corp. for the DAC960E and DAC960P/PD/PD-Ultra/PL. Otherwise, it is strongly recommended that you disable tag queueing to avoid serious problems.

Board Configuration and Auto Configuration
- The SCSI ID of the devices on each channel may not be equal to or greater than the value of the maximum number of targets allowed per channel (MAX_TGT), or it cannot even be configured.

Access to Ready/Standby Drives
- When a SCSI disk drive is initially connected to the controller, it is marked as ready. If a SCSI disk drive is not defined to be part of any physical pack within a system drive at configuration time, it is automatically labeled as a standby drive, which may be used by the controller at any time for automatic failover. For this reason, standby drives are inaccessible from the mlx driver, and the use of ready drives is strongly discouraged. Independent access to ready drives will be removed in an upcoming release.

## FILES
/kernel/drv/mlx.conf mlx configuration file

Last modified 24 Feb 1998
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO

see attributes(5)

WARNINGS

Limitations on SCSI Device Use

Due to Mylex firmware limitations, a tape blocksize greater than 32k bytes cannot be used. Also, tapes and CD-ROM players will not work reliably on channels that also have SCSI hard drives attached to them. Therefore, to be certain of correct SCSI device operation, use SCSI devices only on an otherwise unused channel, and with a fixed block size of 32k or less.

Finally, note that any SCSI command which takes over one hour will automatically be aborted by the Mylex firmware, so very long tape commands (such as erasing a large tape) may fail.

Tag Queueing

Enable tag queueing only for the SCSI disk drives which are officially tested and approved by Mylex Corp. for the DAC960E and DAC960P/PD/PD-Ultra/PL. Otherwise, it is strongly recommended to disable tag queueing to avoid serious problems.

Ready and Standby Drives

If a SCSI disk drive is not defined to be part of any physical pack within a system drive, it is labeled as a ready or standby drive. If any SCSI disk drive within a system drive fails, data on a standby drive may be lost due to the standby replacement procedure. This procedure will overwrite the standby drive if the failed disk drive is configured with any level of redundancy (RAID levels 1, 5, and 6) and its size is identical to the size of the available standby drive.

Therefore, despite the fact that the ready and standby drives are physically connected, the system denies any kind of access to them, so that there will be no chance of accidental loss of valuable data.

Hot Plugging

and DAC960P/PD/PD-Ultra/PL host bus adapter series” Other than the “hot replacement” of disk drives, which is described in the manufacturer’s user’s guide, the Mylex DAC960E series do not support “hot-plugging” (adding or removing devices while the system is running) unless the firmware version of the adapter is 1.22 or 1.23. Otherwise, in order to add or remove devices, you must shut down the system, add or remove devices, reconfigure the host bus adapter using the configuration utility provided by the manufacturer, and then reboot your system.
When setting up the device SCSI target IDs, note that there is a limitation on the choice of target ID numbers. Assuming the maximum number of targets per channel on the particular model of Mylex or IBM host bus adapter is \( \text{MAX\_TGT} \) (see the manufacturer’s user’s manual), the SCSI target IDs on a given channel should range from 0 to \( \text{MAX\_TGT} - 1 \). Note that target SCSI IDs on one channel can be repeated on other channels.

**Example 1:** Mylex DAC960-5 model supports a maximum of four targets per channel, that is, \( \text{MAX\_TGT} = 4 \). Therefore, the SCSI target IDs on a given channel should range from 0 to 3.

**Example 2:** Mylex DAC960-3 model supports a maximum of seven targets per channel, that is, \( \text{MAX\_TGT} = 7 \). Therefore, the SCSI target IDs on a given channel should range from 0 to 6.
NAME | msm – Microsoft Bus Mouse device interface

DESCRIPTION | The msm driver supports the Microsoft Bus Mouse. It allows applications to obtain information about the mouse’s movements and the status of its buttons. The data is read in the Five Byte Packed Binary Format, also called MSC format.

FILES | /dev/msm

ATTRIBUTES | See attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO | attributes(5)
<table>
<thead>
<tr>
<th>NAME</th>
<th>mt – tape interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTION</td>
<td>The files rmt/* refer to tape controllers and associated tape drives. The <code>labelit(1M)</code> command requires these magnetic tape file names to work correctly with the tape controllers. No other tape controller commands require these file names.</td>
</tr>
<tr>
<td>FILES</td>
<td>/dev/rmt/*</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td><code>labelit(1M)</code></td>
</tr>
</tbody>
</table>
mtio(7I) Ioctl Requests

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. EOF 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.

285 SunOS 5.7 Last modified 14 Jan 1997
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(3)` and `aiowrite(3)` 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.

### 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 enabled.
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 MTFSE ioctl can then be 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

write(2) 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.

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/0 is /dev/rmt/0n. 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 \texttt{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 \texttt{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 \texttt{open(2)} and \texttt{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 iocls. The driver returns an ENOTTY error on
unsupported iocls.

The following structure definitions for magnetic tape ioctl commands are from
\texttt{<sys/mtio.h>}.

The minor device byte structure looks as follows:

<table>
<thead>
<tr>
<th>15</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit #</td>
<td>BSD</td>
<td>Reserved</td>
<td>Density</td>
<td>Density</td>
<td>No rewind</td>
<td>Unit #</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

The following operations of MTIOCTOP ioctl 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
- **MREW**: rewind
- **MTOFFL**: rewind and take the drive off-line
- **MNOP**: no operation, sets status only
- **MRETTEN**: 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

Last modified 14 Jan 1997 SunOS 5.7 290
MTGRSZ  get record size

MTLOAD  load the next tape cartridge into the tape drive

/* 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_filename; /* file number of current position */
  daddr_t mtblkno; /* 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 */
};

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.

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 MTERETEN 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 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 mtstruct 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.

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 such as whether the device is SCSI, whether it is a reel device, and whether the device supports absolute file positioning.

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 MTIOCOPERSISTENT 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

Persistent error handling ioctls and asynchronous tape operations

MTIOCOPERSISTENT enables/disables persistent error handling

MTIOCOPERSISTENTSTATUS queries for persistent error handling

MTIOCLRERR clears persistent error handling

MTIOMGUARANTEEDORDER checks whether driver guarantees order of I/O’s

The MTIOCOPERSISTENT 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,

```c
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,

```c
int query;
ioctl(fd, MTIOCPERSISTENTSTATUS, &query);
```

The **MTIOCLRERR** ioctl clears persistent error handling and allows tape operations to continue 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,

```c
ioctl(fd, MTIOCGUARANTEEDORDER)
```

See the Persistent Error Handling subsection above for more information on persistent error handling.

---

**Asynchronous and State Change ioctl**

**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.

```c
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.

- **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,

```c
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,

```c
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 drive.
ioctl Request

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,

```c
ioctl(fd, MTIOCFORCERESERVE);
```

**MTIOCGSHORTFMK**

Enables/disable support for writing short filemarks. This is specific to Exabyte drives.

**MTIOCREADIGNOREILI**

Enables/disable suppress incorrect length indicator support during reads.

**MTIOCREADIGNOREEEOFS**

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 MTIOCGSHORTFMK 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 behavior will be in effect until the device is closed.

For example:

```c
int on = 1;
int off = 0;
/* enable short filemarks */
ioctl(fd, MTIOCGSHORTFMK, &on);
/* disable short filemarks */
ioctl(fd, MTIOCGSHORTFMK, &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 suppressed. The specified tape behavior will be in effect until the device is closed.

For example:

```c
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-recorded-media (EOM). 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 behavior will be in effect until the device is closed.

For example:

```c
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**  Examples of 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:

```sh
mt -F /dev/rmt/0ln bsf 3
mt -F /dev/rmt/0ln fsf 1
dd if=/dev/rmt/0ln
```

To accomplish the same tape positioning in a C program, followed by a get status ioctl:
ioctl Requests

```
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 mt_drivetype mtdt;
struct mtdrivetype_request mtreq;
mtreq.size = sizeof(struct mt_drivetype);
mtreq.mtdtp = &mtdt;
ioctl(fd, MTIOCGETDRIVETYPE, &mtreq);
```

FILES

```
/dev/rmt/<unit number><density>[<BSD behavior>][<no rewind>]
```

- **density**
  - l, m, h, u/c (low, medium, high, ultra/compressed, respectively)
- **BSD behavior (optional)**
  - b
- **no rewind (optional)**
  - 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(3), aiowrite(3), ar(4), st(7D)
- 1/4 Inch Tape Drive Tutorial
**NAME**
ncrets – low-level module for NCR 53C710, 53C810, 53C815, 53C820, and 53C825 host bus adapters

**SYNOPSIS**
ncrets@i0addr, 0

**DESCRIPTION**
The ncrets module provides low-level interface routines between the common disk/tape I/O subsystem and the NCR 53C710, 53C810, 53C815, 53C820, and 53C825 SCSI (Small Computer System Interface) controllers.

The ncrets module can be configured for disk and streaming tape support for one or more host bus adapter boards. Each host bus adapter board 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 it.

The Wide SCSI Bus option (16 bit) on adapters using the 53C820 and 53C825 is not supported. These adapters are operated in their 53C810 compatible mode.

**CONFIGURATION**
The driver attempts to initialize itself in accordance with the information found in the configuration file, ncrets.conf. The relevant user configurable items in this file are as follows:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg</td>
<td>This represents the slot id. The slot id corresponds to the PCI (Peripheral Component Interconnect) hardware slot assigned to the 53C8xx adapter or the EISA base address assigned to the 53C710 adapter.</td>
</tr>
<tr>
<td>reg</td>
<td>reg consists of a 3-tuple of integers. The first integer is a unique identifier for this device. The second and third integer are normally 0. See sysbus(4).</td>
</tr>
<tr>
<td>scsi-initiator-id</td>
<td>The initiator target id is used by the adapter to communicate with devices on the SCSI bus. It must not conflict with any target id value assigned to any disk, tape, or other SCSI devices connected to the adapter. The valid values are 0 to 7. If this property is omitted or if the value isn’t valid, the adapter will use target id 7 by default.</td>
</tr>
<tr>
<td>clock</td>
<td>The synchronous I/O clock frequency. The clock frequency property should match the frequency of the oscillator used to drive the adapter. All the existing implementations use a 40 Mhz clock. However, values between 16 and 75 can be</td>
</tr>
</tbody>
</table>
Devices

specified. If this property is omitted or if the value specified is less than 16 or greater than 75, the SCSI bus will operate in Asynchronous I/O mode.

max-sync-rate

The maximum synchronous I/O transfer rate property can be used to limit the maximum I/O rate for each target device on the SCSI bus. This property consists of a comma separated list of values. The list specifies in order the maximum synchronous I/O rates for each of the targets starting with target 0. The following values are accepted: 10.0, 10, 6.67, 6.66, 5.0, 5, 4.0, 4, 3.33, 3.3, and 0. If any other value is specified or if 0 is specified for a target, the adapter will operate in Asynchronous I/O mode for that particular target.

For example, the entry
max-sync-rate="10,10,10,10,10,10,10"
specifies a maximum synchronous I/O rate of 10 for all targets. This is the default.

no-disconnect

The disable disconnections property controls the disconnection option for each target on the SCSI bus. By default the adapter allows a target to disconnect at any time. If the no-disconnect property is specified, the adapter will not allow the target devices listed to disconnect. Each of the comma separated values specifies a target device for which the option will be disabled. The valid target ids are 0 through 7.

For example, the entry no-disconnect=4,6 disables the disconnect option for target devices 4 and 6.

EXAMPLES

EXAMPLE 1

A sample output.

Here are some examples of entries that could be included in the configuration file, ncrs.conf.

```plaintext
# 53C810 primary controller
# name="ncrs" class="sysbus" reg=11,0,0 ;
```

Last modified 1 Jan 1997

SunOS 5.7
ncrs(7D) Devices

# 53C710 secondary controller, all targets limited to
# 5 MB/sec sync I/O
# name="ncrs" class="eisa" reg=0xc000,0,0
max-sync-rate="5,5,5,5,5,5,5" ;

FILES
/kernel/drv/ncrs.conf configuration file for the ncrs driver

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO
driver.conf(4), sysbus(4), attributes(5)
NAME
nee - Novell NE3200 Ethernet device Driver

SYNOPSIS
#include <sys/stropts.h>
#include <sys/ethernet.h>
#include <sys/dlpi.h>
#include <sys/gld.h>

DESCRIPTION
The nee Ethernet driver is a multi-threaded, loadable, clonable, STREAMS
hardware driver supporting the connectionless Data Link Provider Interface,
dlpi(7P), over Novell NE3200 controllers. Multiple EtherLink 16 controllers
installed within the system are supported by the driver. The nee driver
provides basic support for the NE3200 hardware. Functions include chip
initialization, frame transmit and receive, multicast and “promiscuous”
support, and error recovery and reporting.

APPLICATION PROGRAMMING INTERFACE
The cloning, character-special device /dev/nee is used to access all NE3200
devices installed within the system.

nee and DLPI
The nee driver is dependent on /kernel/misc/gld, a loadable kernel
module that provides the nee 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:

- The maximum SDU is 1500 (ETHERMTU).
- The minimum SDU is 0. The 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

CONFIGURATION
The nee driver does not support the use of shared RAM on the board.
Auto-detect of the media type is also not supported and the board has to be
explicitly configured for which media connector it is using. It is important to
ensure that there are no conflicts for the board’s I/O port, shared RAM, or IRQ level.

**FILES**

- `/dev/nee` nee character special device
- `/kernel/drv/nee.conf` configuration file of nee driver

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

**SEE ALSO**

attributes(5), dlpi(7P), gld(7D)
NAME
nei – Novell NE2000, NE2000plus Ethernet device Driver

SYNOPSIS
/dev/nei

DESCRIPTION
The nei Ethernet driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface, dlpi(7P), over Novell NE2000 and NE2000plus controllers. The nei driver provides basic support for the NE2000 and NE2000plus hardware. Functions include chip initialization, frame transmit and receive, multicast and “promiscuous” support, and error recovery and reporting. Multiple NE2000 and NE2000plus controllers installed within the system are supported by the driver.

APPLICATION PROGRAMMING INTERFACE
The cloning, character-special device /dev/nei is used to access all NE2000 series devices installed within the system.

nei and DLPI
The nei driver is dependent on /kernel/misc/gld, a loadable kernel module that provides the nei 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:

- The maximum Service Data Unit (SDU) is 1500 (ETHERMTU - defined in <sys/ethernet.h>).
- The minimum SDU is 0. The driver will pad to the mandatory 60-octet minimum packet size.
- The dlsap address length is 8.
- The Media Access Control (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 nei driver supports the use of shared RAM only on NE2000plus boards. It is important to ensure that there are no conflicts for the board’s I/O port, shared RAM, or IRQ level. See the Solaris 2.4 x86 Driver Update Guide for additional restrictions on configuring the NE2000 and other Ethernet cards in the same system.

FILES
/dev/nei
nei character special device

Last modified 1 Jan 1997
SunOS 5.7
304
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO

`attributes(5)`, `dlpi(7P)`, `gld(7D)`, `streamio(7I)`
**NAME**  
nfe – Compaq Netflex-2 Dualport Ethernet and ENET/TR Drivers

**SYNOPSIS**  
/dev/nfe

**DESCRIPTION**  
The nfe Ethernet driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface, **dlpi**(7P), over Compaq Netflex-2 controllers. Multiple Netflex-2 controllers installed within the system are supported by the driver. The nfe driver provides basic support for the Netflex-2 hardware. Functions include chip initialization, frame transmit and receive, multicast support, and error recovery and reporting. NOTE: Promiscuous mode is not supported using the current revision of the firmware.

**APPLICATION PROGRAMMING INTERFACE**  
The cloning, character-special device /dev/nfe is used to access all Netflex-2 devices installed within the system.

**nfe and DLPI**  
The nfe driver is dependent on /kernel/misc/gld, a loadable kernel module that provides the nfe 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 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 service mode is DL_CLDLS.
- The broadcast address value is Ethernet/IEEE broadcast address (FF:FF:FF:FF:FF:FF).

**CONFIGURATION**  
The nfe driver does not support the use of shared RAM on the board. Auto-detect of the media type is also not supported and the board has to be explicitly configured for which media connector it is using.

**FILES**  
/dev/nfe  
nfe character special device

Last modified 16 May 1997
/kernel/drv/nfe.conf configuration file of nfe driver

<sys/stropts.h>
<sys/ethernet.h>
<sys/dlpi.h>
<sys/gld.h>

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO attributes(5), dlpi(7P), gld(7D)
null – the null file

/dev/null

Data written on the null special file, /dev/null, is discarded.
Reads from a null special file always return 0 bytes.

/dev/null
NAME  openprom – PROM monitor configuration interface

SYNOPSIS

#include <sys/fcntl.h>

#include <sys/types.h>

#include <sys/openpromio.h>

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 {
    unit_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.

IOCTLS

OPROMGETOPT

This ioctl takes the null-terminated name of a property in the oprom_array
and returns its null-terminated value (overlapping 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

309  SunOS 5.7  Last modified 13 Jan 1997
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, and 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` routine.

**EINVAL** The size value was invalid, or (for `OPROMSETOPT`) the property does not exist, or and invalid ioctl is being issued.

**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  A sample program.

The following example shows how the `oprom_array` is allocated and reused for data returned by the driver.

```c
/*
 * 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 BUF SZ 1024 /* A handy default buffer size */
#define MAXVALSZ (BUFSZ - sizeof (int))
static char *promdev = "/dev/openprom";
/*
 * Allocate an openpromio structure big enough to contain
 * a bufsiz ed `oprom_array`. Zero out the structure and
 * set the `oprom_size` field to bufsiz e.
 */
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.
 */
static int
peer(int nodeid, int fd)
```
Devices

openprom(7D)

{  
  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
    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>
", 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 OPROMSETOPT should be performed after an OPROMGETOPT 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
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:

<table>
<thead>
<tr>
<th>Platform</th>
<th>/devices name</th>
<th>/dev/dsk name</th>
</tr>
</thead>
<tbody>
<tr>
<td>x86</td>
<td>/devices/isa/pcic@1,3e0/disk@0:a</td>
<td>/dev/dsk/c1d0s0</td>
</tr>
<tr>
<td>SPARC</td>
<td>.nf /devices/iommu@f,e0000000/sbus@f,e0001000/SUNW,pcmcia@3,0/disk@0:a</td>
<td>/dev/dsk/c1d0s0</td>
</tr>
</tbody>
</table>

FILES
/kernel/drv/pcata pcata driver

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:
Devices

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWpsdpr</td>
</tr>
</tbody>
</table>

SEE ALSO: format(1M), mount(1M), newfs(1M), pcmcia(4), attributes(5), pcfs(7FS)
NAME       pcelx – 3COM EtherLink III PCMCIA Ethernet Adapter
SYNOPSIS   \texttt{network@<socket>:\texttt{pcelx}<socket>}
DESCRIPTION The \texttt{pcelx} driver supports the 3COM EtherLink III PCMCIA PC Card as a standard Ethernet type of device conforming to the DLPI interface specification. The driver supports the \textit{hot-plugging} of the PC Card.

The PPA (Physical Point of Attachment) is defined by the socket number the PC Card is inserted in. This means that for IP use, the PC Card should always be plugged into the same socket that the network interface was initially brought up on or else a network reconfiguration should be done to take down the old interface and bring up the new one.

The 3C589, 3C589B, and 3C589C versions of the PC Card are supported on the x86 platform. The 3C589B and 3C589C are supported on the SPARC platform.

FILES
\begin{itemize}
  \item \texttt{/kernel/drv/pcelx}         \texttt{pcelx} driver
  \item \texttt{/dev/pcelx}                    DLPI Style 2 device
  \item \texttt{/dev/pcelx}n                    DLPI Style 1 device where: \texttt{n} is the PCMCIA physical socket number.
\end{itemize}

SEE ALSO \texttt{pcmcia(4)}

Last modified 18 Feb 1997
pcfs (7FS) File Systems

NAME
cvfs – DOS 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", struct pcfs_args, struct *pc_argp, sizeof (struct pcfs_args));

DESCRIPTION
cvfs is a file system type that allows users direct access to files on DOS formatted disks from within the SunOS operating system. Once mounted, cvfs provides standard SunOS file operations and semantics. That is, users can create, delete, read, and write files on a DOS formatted disk. They can also create and delete directories and list files in a directory.

The cvfs filesystem 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. When mounting a cvfs filesystem, a pointer to a structure containing mount flags and local timezone information, *pc_argp, is required:

struct pcfs_args {
    int timezone; /* seconds west of Greenwich */
    int daylight; /* type of dst correction */
    int flags;  }
>

The information required in the timezone and daylight fields in this structure is described in ctime(3C). flags can contain the following flag:

PCFS_MNT_FOLDCASE
Fold names read from the filesystem to lowercase.

Mounting File Systems
cvfs is mounted from diskette with the command:
    mount -F cvfs device-special directory-name
or you can use:
    mount directory-name
if the following line is in your /etc/vfstab file:
    device-special directory-name cvfs no rw
x86: cvfs is mounted from the hard disk with the command:
    mount -F cvfs
device-special:logical-drive directory-name
or you can use:
    mount directory-name
if the following line is in your /etc/vfstab file:
    device-special:logical_drive directory-name
cvfs no rw
device-special specifies the special block device file for the diskette (/dev/disketteN) or the entire hard disk (/dev/dsk/cNdNp0 for a
SCSI disk, and /dev/dsk/cNdNp0 for IDE disks) or the PCMCIA pseudo-floppy memory card (/dev/dsk/cNtNdNsN).

On x86 systems, 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 DOS 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, on x86, to mount the Primary DOS partition from a SCSI hard disk, use:

```
mount -F pcfs /dev/dsk/cNdNdNp0:c
```

On x86, to mount the first logical drive in the Extended DOS partition from an IDE hard disk, use:

```
mount -F pcfs /dev/dsk/cNdNdNd0:d
```

To mount a DOS diskette in the first floppy drive, if Volume Management is not running (see vold(1M)) use:

```
mount -F pcfs /dev/diskette /pcfs/a
```

If Volume Management is running, then running volcheck(1) will automatically mount the floppy and some removable disks for the user.

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/cNdNdNsN /pcfs
```

**Conventions**

Files and directories created through pcfs have to comply with either the DOS short filename convention or the long filename convention introduced with Windows 95. The DOS short filename 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 filename convention is much closer to Solaris filenames. A long filename can consist of any any characters valid in a short filename, lowercase letters, non-leading spaces, the characters +, ; = [ ], any number of periods, and be up to 255 characters long. Long filenames have an associated short filename for systems that do not support long filenames (such as earlier releases of Solaris). The short filename is not visible if the system recognizes long filenames. pcfs generates a unique short name automatically when creating a long filename. Given a long filename such as `This is a really long filename.TXT`, the short filename will generally be of the form `THISIS~N.TXT`, where N is a number. So, this long filename 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 need to use \texttt{pcfs} filesystems on systems that do not support long filenames, you may want to continue following the short filename conventions. See \texttt{EXAMPLES}.

When creating a filename, \texttt{pcfs} creates a short filename if it fits the DOS short filename format, otherwise it creates a long filename. This is because long filenames take more directory space. In fact, since the root directory of a \texttt{pcfs} filesystem is fixed size, long filenames in the root directory should be avoided if possible.

When displaying filenames, \texttt{pcfs} shows them exactly as they are on the media (so short names show up as all uppercase, and long filenames retain their case). The old behavior of \texttt{pcfs} was to fold all names to lowercase, which can be forced with the \texttt{PCFS\_MNT\_FOLDCASE} mount option. All filename searches within \texttt{pcfs}, however, are treated as if they were uppercase, so \texttt{readme.txt} and \texttt{ReAdMe.TxT} refer to the same file.

One can use either the DOS \texttt{FORMAT} command, or the command:
\begin{verbatim}
fformat -d
\end{verbatim}
in the SunOS system to format a diskette or a PCMCIA pseudo-floppy memory card in DOS format.

\noindent \textbf{Boot Partitions}

On x86 systems, hard drives may contain an \texttt{fdisk} partition reserved for the Solaris boot utilities. These partitions are special instances of \texttt{pcfs}. An x86 boot partition may be mounted with the command:
\begin{verbatim}
mount -F pcfs
device-special:boot directory-name
\end{verbatim}
or you can use:
\begin{verbatim}
mount directory-name
\end{verbatim}
if the following line is in your \texttt{/etc/vfstab} file:
\begin{verbatim}
device-special:boot - directory-name
pcfs - no rw
\end{verbatim}
\texttt{device-special} specifies the special block device file for the entire hard disk (/dev/dsk/CNtNdNp0)
\texttt{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 user may create, delete, or modify files on a boot partition.

\noindent \textbf{EXAMPLES}

\noindent \textbf{EXAMPLE 1} \hspace{1cm} Sample displays of filenames.

If you copy a file:
\begin{verbatim}
financial.data
\end{verbatim}
from a UNIX file system to \texttt{pcfs}, it will show up as:
\begin{verbatim}
financial.data,
\end{verbatim}
in \texttt{pcfs}, but will probably show up as
\begin{verbatim}
FINANC-1.DAT
\end{verbatim}
in systems that do not support long filenames.
The following file names:

```plaintext
test.sh.orig
data+.login
```

are not legal short filenames, but are legal long filenames. Other systems that do not support long filenames may well see:

```plaintext
TESTSH~1.ORI
DATA~1
LOGIN~1
```

It is also a good idea to keep in mind that the short filename is generated from the initial characters of the long filename, so it is better to differentiate names in the first few characters. As an example, these names:

```plaintext
WorkReport.February.Data
WorkReport.March.Data
```

result in these short names, which are not very distinguishable:

```plaintext
WORKRE~1.DAT
WORKRE~2.DAT
WORKRE~3.DAT
```

These names, however:

```plaintext
January.WorkReport.Data
February.WorkReport.Data
March.WorkReport.Data
```

result in the more descriptive short names:

```plaintext
JANUAR~1.DAT
FEBRUA~1.DAT
MARCHW~1.DAT
```

FILES

```
/usr/lib/fs/pcfs/mount
/usr/kernel/fs/pcfs
```

SEE ALSO

```
chgrp(1), chown(1), dos2unix(1), eject(1), fdformat(1), unix2dos(1),
doscheck(1), mount(1M), mount_pcfs(1M), vold(1M), ctime(3C),
vfstab(4), pcmem(7D)
```

x86 Only

```
fdisk(1M)
```

WARNINGS

Physically ejecting a DOS floppy while the device is still mounted as pcfs is not recommended. In addition, if Volume Management is managing a device, the `eject(1)` command should be used before physically removing media.

x86: When mounting pcfs on a hard disk, the first block on that device must contain a valid fdisk partition table.

pcfs has no provision for handling owner-IDs or group-IDs on files. You may experience various errors coming from `chown(1)` or `chgrp(1)`. This is not a problem, it is a limitation of pcfs.
The following are all the legal characters that are allowed in short file names (or their extensions) in `pcfs`:

`0-9, A-Z, and $@%^()_<>`~`

Since SunOS and DOS operating systems use different character sets, and have different requirements for the text file format, one can use the `dos2unix(1)` or `unix2dos(1)` commands to convert files between them.

`pcfs` offers a convenient transportation vehicle for files between Sun Workstations and PCs. Since the DOS disk format was designed for use under DOS, it is quite inefficient to operate under the SunOS system. Therefore, it should not be used as the format for a regular local storage. You should use `ufs` for local storage within the SunOS system.

Though long filenames can contain spaces (just as in UNIX filenames), some utilities may be confused by them.

This implementation of `pcfs` conforms to the behavior exhibited by Windows 95 version 4.00.950.

`pcfs` should handle the disk change condition in the same way that DOS does, so that the user does not need to unmount the file system to change floppies.

`pcfs` does not include files with the `hidden` or `system` bits set when listing or searching a directory.
<table>
<thead>
<tr>
<th>NAME</th>
<th>pcic – Intel i82365SL PC Card Interface Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTION</td>
<td>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. The chips that have been tested are:</td>
</tr>
<tr>
<td></td>
<td>- Intel 82365SL</td>
</tr>
<tr>
<td></td>
<td>- Cirrus Logic PD6710/PD6720/PD6722</td>
</tr>
<tr>
<td></td>
<td>- Vadem VG365/VG465/VG468/VG469</td>
</tr>
<tr>
<td></td>
<td>- Toshiba PCIC and ToPIC</td>
</tr>
<tr>
<td></td>
<td>- Ricoh RF5C366</td>
</tr>
<tr>
<td></td>
<td>- Texas Instruments PCI1130/PCI1131/PCI1031</td>
</tr>
<tr>
<td></td>
<td>While most systems using one of these chips should work, there are enough options left to the hardware designer that are not software detectable that some systems will not be supported. Note that systems with CardBus interfaces are only supported in the non-legacy mode. Systems that only initialize the bridge to legacy mode and don’t configure the PCI memory will not be supported. Direct access to the PCMCIA hardware is not supported. All device access must be through the Card Services interface of the DDI.</td>
</tr>
<tr>
<td>CONFIGURATION</td>
<td>There is one driver configuration property defined in the pcic.conf file.</td>
</tr>
<tr>
<td>Driver Configuration</td>
<td>interrupt-priorities=11; This property must be defined and must not be modified from the default value.</td>
</tr>
<tr>
<td>FILES</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/kernel/drv/pcic pcic driver</td>
</tr>
<tr>
<td></td>
<td>/kernel/drv/pcic.conf pcic configuration file</td>
</tr>
<tr>
<td>SEE ALSO</td>
<td>pcmcia(4) and stp4020(7D)</td>
</tr>
</tbody>
</table>

Last modified 20 Mar 1997 SunOS 5.7 322
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), lterm(7M), pterm(7M), streamio(7I), termio(7I)

STREAMS   Programming  Guide
NAME
pcmem – PCMCIA memory card nexus driver

DESCRIPTION
The pcmem driver identifies the type of memory card in the system and will allow future support of other memory device types.

The PCMCIA memory card nexus driver supports PCMCIA memory card client drivers. There are no user-configurable options for this driver.

FILES
/kernel/drv/pcmem       pcmem driver

SEE ALSO
pcram(7D)
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
pcn and DLPI

The cloning character-special device, /dev/pcn, is used to access all PCnet
devices installed in the system.

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
therefor 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 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 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 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.

FILES

/dev/pcn character special device
/kernel/drv/pcn.conf configuration file

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO

attributes(5), standards(5), dlpi(7P), streamio(7I)

Writing Device Drivers
STREAMS Programming Guide
NAME
pcram – PCMCIA RAM memory card device driver

SYNOPSIS
memory@<socket>/pcram@<technology>,0:c
memory@<socket>/pcram@<technology>,0:c,raw

DESCRIPTION
The PCMCIA RAM memory card device driver supports disk-like I/O access to any standard PCMCIA static random access memory (SRAM) card and dynamic random access memory (DRAM) card. The driver supports standard PCMCIA SRAM/DRAM cards that contain a Card Information Structure (CIS). RAM card densities in the 512Kilobytes to 64Mbyte range are supported.

FILES
/kernel/drv/pcram  pcram driver
/dev/dsk/cntndnsn  block files
/dev/rdsk/cntndnsn  raw files

where:
 cn    controller n
 tn    technology type n
 0x1ROM,0x2OTPROM,0x3EPROM,
 0x4EEPROM,0x5FLASH,0x6SRAM,
 0x7DRAM
 dn    technology region in type n
 sn    slice n

SEE ALSO
fdformat(1), pcmcia(4), dkio(7I), pcmem(7D)
NAME pcscsi – low-level module for the AMD PCscsi, PCscsi II, PCnet-SCSI, and Qlogic QLA510 PCI-to-SCSI bus adapters

SYNOPSIS pcscsi@ioaddr, 0

DESCRIPTION The pcscsi module provides low-level interface routines between the common disk/tape I/O subsystem and the Am53C974 (PCscsi), Am53C974A (PCscsi II), Am79C974 (PCnet-SCSI) (SCSI device only), and the Qlogic QLA510 Small Computer System Interface (SCSI) controllers.

The pcscsi module can be configured for disk and streaming tape support for one host bus adapter device. Each host bus adapter device must be the sole initiator on a SCSI bus. Auto-configuration code determines if the adapter is present on the PCI bus, what its configuration is, and what types of devices are attached to it.

Because these are PCI devices, any configuration is done through the PCI BIOS. Generally these settings can be accessed through a CMOS utility.

CONFIGURATION The driver attempts to initialize itself in accordance with the configuration the PCI BIOS assigned to the chip.

While there is information found in the configuration file, pcscsi.conf, this information is used only by the I/O subsystem. There are no user-configurable options.

FILES

/kernel/drv/pcscsi.conf configuration file for the pcscsi driver

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO driver.conf(4), sysbus(4), attributes(5)
NAME  pcser – PCMCIA serial card device driver

SYNOPSIS

serial@<socket>:pcser

serial@<socket>:pcser, cu

DESCRIPTION

The 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 presents an 8250-type UART interface.

FILES

/kernel/drv/pcser  pcser driver
/dev/term/pcnn  dial-in devices
/dev/cua/pcnn  dial-out devices where: n is the PCMCIA physical
socket number.

SEE ALSO

cu(1C), tip(1), uucp(1C), autopush(1M), pcmcia(1M), ports(1M),
ioctl(2), open(2), pcmcia(4), ldterm(7M), termio(7I ), 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
pe – Xircom Pocket Ethernet device driver

SYNOPSIS
#include <sys/stropts.h>
#include <sys/ethernet.h>
#include <sys/dlpi.h>
#include <sys/gld.h>

DESCRIPTION
The Xircom Pocket Ethernet driver (pe) is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface, dlpi(7P), with a Xircom Pocket Ethernet Adapter III (PE3). Multiple PE3 controllers installed within the system are supported by the driver.

PE and DLPI
The pe driver provides basic support for the PE3 hardware. Functions include chip initialization, frame transmission and reception, multicast and "promiscuous" support, and error recovery and reporting.

The pe driver supports both bi-directional and unidirectional parallel ports. The Port and Adapter type is automatically detected and set when the driver initializes.

It is important not to attempt to use any other driver that may also use the same port when the pe driver is operational. This may interfere with network traffic that is currently being sent or received by the PE3 adapter.

The cloning, character-special device /dev/pe is used to access all PE3 controllers installed within the system.

The pe driver is dependent on /kernel/misc/gld, a loadable kernel module that provides the pe 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:

- The max SDU is 1500 (ETHERMTU).
- The min SDU is 0.
- The dlsap address length is 8.
- The MAC type is DL_ETHER or DL_CSMACD.
- 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/pe      device file

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`attributes(5)`, `dlpi(7P)`
NAME
pfmod – STREAMS Packet Filter Module

SYNOPSIS
#include <sys/pfmod.h>

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. IOCTLs 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:

```c
struct packetfilt {
    uchar_t Pf_Priority; /* priority of filter */
    uchar_t Pf_PriorityLen; /* length of filter cmd list */
    ushort_t Pf_Filter[ENMAXFILTERS]; /* filter command list */
};
```

Last modified 18 Sep 1992
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_t), 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_t ("shortwords"). Each shortword of the command list specifies one of the actions ENF_PUSHLIT, ENF_PUSHZERO, ENF_PUSHONE, ENF_PUSHFFFF, ENF_PUSHFF00, ENF_PUSH00FF, or ENF_PUSHWORD+n, which respectively push the next shortword of the command list, zero, one, 0xFFFF, 0xFF00, 0x00FF, or shortword n of the subject message on the stack, and a binary operator from the set [ENF_EQ, ENF_NEQ, ENF_LT, ENF_LE, ENF_GT, ENF_GE, ENF_AND, ENF_OR, ENF_XOR] which then operates on the top two elements of the stack and replaces 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**

**EXAMPLE 1** 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.

```c
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)
sysex("pfmod");

/*
 * Set up filter. Offset is the displacement of the Ethernet type field from
 * the beginning of the packet in units of ushort_t's.
 */
offset = ((uint_t) &eh.ether_type - (uint_t) &eh.ether_dhost) /
sizeof (us_short);
*fwp++ = ENF_PUSHWORD + offset;
*fwp++ = ENF_PUSHLIT;
*fwp++ = htons(ETHERTYPE_REVARP);
*fwp++ = ENF_EQ;
pf.Pf_FilterLen = fwp - &pf.Pf_Filter[0];
```

**EXAMPLE 2** This filter can be abbreviated by taking advantage of the ability to combine actions and operations:

```c
*fwp++ = ENF_PUSHWORD + offset;
*fwp++ = ENF_PUSHLIT | ENF_EQ;
*fwp++ = htons(ETHERTYPE_REVARP);
```
SEE ALSO

bufmod(7M), dlpi(7P), ie(7D), le(7D), pullupmsg(9F)
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

Last modified 21 Aug 1992

SunOS 5.7

336
NAME       pln – SPARCstorage Array SCSI Host Bus Adapter Driver

SYNOPSIS   pln@SUNW,pln@a0000800,200611b9

DESCRIPTION The pln Host Bus Adapter (HBA) driver is a SCSA compliant nexus driver which supports the SPARC Storage Array. The SPARC Storage Array is a disk array device which supports multiple disk drives. The drives are located on several SCSI busses within the SPARC Storage Array. A SPARC microprocessor controls the SPARC Storage Array. Non-volatile RAM is used as a disk cache. The SPARC Storage Array interfaces to the host system using Fibre Channel. An SBus card called the SOC card (see soc(7D)) connects the Fibre Channel to the host system.

The pln driver interfaces with the SOC device driver, soc(7D), and the SPARC Storage Array SCSI target driver, ssd(7D).

The pln driver supports the standard functions provided by the SCSA interface. The driver supports tagged and untagged queuing and auto request sense.

FILES       /kernel/drv/pln          ELF kernel module
             /kernel/drv/pln.conf     configuration file

SEE ALSO   prtconf(1M), ssaadm(1M), driver.conf(4), soc(7D), ssd(7D)

Writing Device Drivers

ANSI Small Computer System Interface-2 (SCSI-2)

DIAGNOSTICS The messages described below may appear on the system console and in the system log.

This following messages indicate the pln driver was unable to attach to the device. These messages are preceded by "pln%d", where "%d" is the instance number of the pln controller.

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.
Driver was unable to attach to the hardware for some reason that may be printed. SCSI devices will be inaccessible.
NAME
pm – power management driver

SYNOPSIS
/dev/pm

DESCRIPTION
The Power Management 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 pm model of power management is to view the system as a collection of
devices. Each device is a collection of components. A component is the
smallest power manageable unit. The definition of power manageable
components of a device is under the control of the device driver. A power
manageable component has three states. It may be busy (in use), it may be idle
(not in use but using normal power), or it may be power managed (not in use
and not using normal power).

Normally, the pm driver manages the component transition from idle to
power managed. pm 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. See power.conf(4),
power(9E), pm_create_components(9F), pm_idle_component(9F),
pm_busy_component(9F), pm_destroy_components(9F), and Writing
Device Drivers for additional information.

Using the commands below, an application may take control of the power
management of a device from the pm driver and manage the transition of
device power levels directly. Xsun(1) does this to power manage the display.

IOCTLS
For all ioctl commands, arg (see ioctl(2)) points to a structure of type
pm_request defined in <sys/pm.h>:

typedef struct {
    char *who; /* device to configure */
    int select; /* selects the component or
dependent of the device */
    int level; /* power level or threshold value */
    char *dependent; /* hold name of dependent */
    int size; /* size of dependent buffer */
} pm_request;
The fields should contain the following data:

- **who**: Pointer to the name of the device to be configured. The name must be in the format described in `power.conf(4)`.

- **select**: Non-negative integer specifying the component or dependent being configured. The numbering starts at zero.

- **level**: Non-negative integer specifying the threshold value in seconds or the desired power level.

- **dependent**: Pointer to a buffer which contains or receives the name of a device on which this device has a dependency. It uses the same format as the `who` field.

- **size**: Size of the dependent buffer.

Not all fields are used in each command.

### PM_DISABLE_AUTOPM

The device named by `who` is disabled from being automatically power managed. The caller will power manage the device directly using the commands below. If this command is not successfully executed, subsequent `PM_SET_CUR_PWR` calls will fail. Error codes:

- **EBUSY**: Device already disabled for auto power management.
- **EPERM**: Caller is neither superuser nor owner of the device.

### PM_GET_NORM_PWR

The normal power level of the component `select` of the device named by `who` is returned. The normal power level of the component is the power level to which the component will be set when it becomes busy again. Error codes:

- **EINVAL**: Device component out of range.
- **EIO**: Device has no power-manageable components.

### PM_GET_CUR_PWR

The current power level of component `select` of the device named by `who` is returned. Error codes:

- **EINVAL**: Device component out of range.
**PM_SET_CUR_PWR**

Component select of the device named by *who* is brought to power level *level*. If select is not 0 and component 0 of the device is at power level 0, component 0 is brought to its normal power level. Each component of each device which depends on this device is brought to its normal power level. Each component of each ancestor of each device affected is brought to its normal power level. Error codes:

- **EINVAL** Device component out of range, or power level < 0.
- **EIO** Failed to power device or its ancestors or its dependees or their ancestors.
- **EPERM** Caller is neither superuser nor owner of the device.

**PM_REENABLE_AUTOPM**

The device named by *who* is re-enabled for automatic power management. By default, all configured devices are automatically power managed by the `pm` device. Error codes:

- **EINVAL** Device already being power managed automatically.
- **EPERM** Caller is neither superuser nor owner of the device.

### 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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Stability</td>
<td>Unstable</td>
</tr>
</tbody>
</table>
SEE ALSO  

pmconfig(1M), intro(2), ioctl(2), power.conf(4), attributes(5),  
attach(9E), detach(9E), power(9E), ddi_dev_is_needed(9F),  
pm_busy_component(9F), pm_create_components(9F),  
pm_destroy_components(9F), pm_get_normal_power(9F),  
pm_idle_component(9F), pm_set_normal_power(9F)  

Writing Device Drivers

NOTES  

To unload a power managed driver, the driver must first be unmanaged (see pmconfig(1M)).

pm(7D) is not a stable API. The commands documented within this reference manual page may not exist or may have a different interpretation in a future release. That other reference manual pages lack this notice does not imply that they document a stable API.
NAME  

pmc – Platform Management Chip driver

SYNOPSIS  

SUNW, pmc@slot, offset: pmc

DESCRIPTION  

The Platform Management Chip driver provides a number of miscellaneous platform-specific functions. These functions provide power control for devices that cannot manage their own power control and provide information about the connection status of the machine. Not all functions are supported on all platforms.

The user interface is provided through ioctl(2) commands. The pmc driver can be accessed using /dev/pmc. The pmc device is accessed by the system using the platform-pm property of the root node.

IOCTLS  

These ioctl requests fall into three categories: connection status, power control, and miscellaneous. The connection status can be used to determine whether the keyboard, ethernet, and ISDN devices are plugged in. The power control function controls the removal of the platform power. Miscellaneous functions enable reading of the digital to analog converter.

The following ioctl(2) requests require only an open file descriptor for the pmc device and an ioctl command. The arg argument is not used. Refer to ioctl(2) for more information. The following pmc ioctl commands are defined in <sys/pmcio.h>.

PMC_GET_KBD  

This command returns the connection status of the keyboard. If the keyboard is connected PMC_KBD_STAT is returned; otherwise, 0 is returned.

PMC_GET_ENET  

This command returns the connection status of the ethernet. If the ethernet is connected PMC_ENET_STAT is returned; otherwise, 0 is returned.

PMC_GET_ISDN  

This command returns the connection status of the ISDN channels. The return value is a bit map of the connected channels: PMC_ISDN_ST0 for NT and PMC_ISDN_ST1 for TE.

PMC_GET_A2D  

This command returns the result of an 8-bit analog-to-digital conversion. The meaning of the returned data is platform-specific.

PMC_POWER_OFF  

This command is available only to the super-user. It turns off all power to the system. Note that critical data may be lost if proper preparation prior to power removal is not performed.

The poll(2) interface is supported. It can be used to poll for connection status changes. A process wishing to detect such connection changes should use the POLLIN event flag. When any connection status changes, the poll(2)
mechanism will be notified. It is up to the user to verify whether the connection status change is of interest.

**ERRORS**

**EPERM** Must be privileged user to use PMC_POWER_OFF.

**FILES**

/dev/pmc power management chip device

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>SUN4m, SPARCstation Voyager</td>
</tr>
</tbody>
</table>

**SEE ALSO**

intro(2), ioctl(2), open(2), poll(2), attributes(5), pm(7D)
ppp is a STREAMS module which implements the Point to Point Protocol (PPP). PPP is a datalink protocol which provides a method for transmitting datagrams over serial point-to-point links. PPP allows for various options to be negotiated between the two hosts of a point-to-point link; these options provide things such as peer authentication, header compression, link quality monitoring, and mapping of control characters. The PPP specifications are described in RFC 1331 The Point-to-Point Protocol (PPP) for the Transmission of Multi-protocol Datagrams over Point-to-Point Links and RFC 1332 The PPP Internet Protocol Control Protocol (IPCP).

The pseudo device drivers /dev/ipd, /dev/ipdptp, and /dev/ipdcm form the IP-dialup layer. This layer provides IP network interfaces for dialup (connect on demand) point-to-point links. The ipd and ipdptp devices are the IP-dialup network interfaces. The ipd device provides a point-to-multipoint interface, and the ipdptp device provides a point-to-point interface. The ipdcm device supplies an interface between the ipd or ipdptp device and a link manager.

The ppp module and IP-dialup layer work together to provide IP connectivity over serial point-to-point links. A “link manager” daemon is responsible for setting up and tearing down these dialup connections. Connections are established when an IP packet needs to be sent to the remote host, or the remote host has indicated its desire to establish a PPP connection.

The ppp_diag module captures PPP layer packets and parses the contents for debugging purposes. Usually, the parsed output is sent to the strlog facility from which it is retrieved by the link manager. This module is pushed between the serial device and the ppp module by the link manager when debugging is enabled.

When a packet is routed to an IP-dialup point-to-point interface which is not currently connected to the remote host, the ipdcm driver sends a message to the link manager to establish the connection. The link manager opens a communications channel and pushes the ppp module onto the corresponding serial device. The ppp module negotiates with the remote host on which options will be used for the link. When both hosts have agreed on a set of options, the link manager links the ppp module and serial device underneath the ipd or ipdptp interface which is providing the IP interface to the remote host.

Similarly, a remote host may initiate a connection on an enabled communications port. In this case the link manager receives the request and pushes the ppp module onto the corresponding device. Once the ppp module has successfully negotiated on the set of options for the link with its peer, the
link manager links the ppp module and serial device underneath the ipd or ipdptp interface which is providing the IP-dialup interface.

When the ppp module and serial device have been linked underneath the IP-dialup interface, IP packets are sent and received over the point-to-point link in PPP frames.

FILES

/dev/ipd  pseudo device driver that provides point-to-point interface.
/dev/ipdptp  pseudo device driver that provides point-to-multipoint interface.
/dev/ipdcm  pseudo device driver that provides interface between ipd and ipdptp and link manager.

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWpppk</td>
</tr>
</tbody>
</table>

SEE ALSO  aspppd(1M), attributes(5)
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, TIOCWSIN, and TIOCWSIN 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 major/minor number of the master device which can be used to determine the name of the corresponding slave device. 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 endpoints 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.

Last modified 21 Aug 1992

SunOS 5.7

350
EXAMPLE 1  A sample program.

```c
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 UART).

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 ioctl's 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 \(\text{VSTART}\) member of the \(\text{c}_\text{cc}\) 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 \texttt{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**

**EXAMPLE 1** A sample program.

```c
#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.
### Devices

#### NAME

qe – QEC/MACE Ethernet device driver

#### SYNOPSIS

```c
#include <mace.h>

#include <qe.h>

#include <qec.h>

#include <dlpi.h>
```

#### DESCRIPTION

qe is a multi-threaded, loadable, clonable, STREAMS hardware device driver supporting the connectionless Data Link Provider Interface, `dlpi(7P)`, over Am79C940 (MACE) Ethernet controllers in the SBus QED card. qec(7D) is its parent in the Open Boot Prom device tree. There is no fixed limitation on the number of QED cards supported by the driver. The qe driver provides basic support for the MACE and QEC hardware. Functions include chip initialization, frame transmit and receive, multicast and promiscuous support, and error recovery and reporting.

The cloning character-special device `/dev/qe` is used to access all MACE controllers installed within the system.

#### qe and DLPI

The qe driver is a “style 2” Data Link Service provider. All `M_PROTO` and `M_PCPROTO` type msgs 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 device is initialized on first attach and de-initialized (stopped) on 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 max SDU is 1500 (`ETHERMTU`).
- The min 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.

---

Last modified 3 Mar 1994

SunOS 5.7

356
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 (0xFFFFFFFF).

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 **qe** 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 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 along with a proper ether header is expected as part of the data.

The **qe** 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_BOUNDED** state, the user may transmit frames on the Ethernet by sending **DL_UNITDATA_REQ** messages to the **qe** driver. The **qe** 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.

**qe Primitives**

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 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 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 return 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 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 on this stream is successful. Once changed, all streams subsequently opened and attached to this device will obtain this new physical address. Once changed, the physical address will remain so until this primitive is used to change the physical address again or the system is rebooted, whichever comes first.

**FILES**

/dev/qe qe special character device.

**SEE ALSO**

dlpi(7P), ie(7D), le(7D), qec(7D)

Last modified 3 Mar 1994 SunOS 5.7 358
NAME
qe – QEC bus nexus device driver

DESCRIPTION
The qe device driver is a bus nexus driver which provides basic support for the QEC hardware. It is the parent of the qe(7D) leaf driver. The driver supports multiple QED SBus cards installed within the system. It is not directly accessible to the user.

SEE ALSO qe(7D)
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 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 and 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. 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>).

Last modified 6 May 1998
The minimum SDU is 0.

The dlsap address length is 8.

The MAC type is DL_EETHER.

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 (0xFFFFFFF).

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. 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 or 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 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 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.

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 4 following mode:

- 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 Mbps and accordingly, IPG will be set to 9.6 or 0.96
microseconds.

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 and 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/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), ieee(7D), le(7D)
NAME
quoteactl – manipulate disk quotas

SYNOPSIS
#include <sys/fs/ufs_quota.h>

int ioctl(int fd, int Q_QUOTACTL, struct quotctl *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.
ioctl Requests

**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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>on success.</td>
</tr>
<tr>
<td>-1</td>
<td>on failure and sets <em>errno</em> to indicate the error.</td>
</tr>
</tbody>
</table>

**ERRORS**  

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EFAULT</td>
<td><em>addr</em> is invalid.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>The kernel has not been compiled with the QUOTA option. <em>op</em> is invalid.</td>
</tr>
<tr>
<td>ENOENT</td>
<td>The quotas file specified by <em>addr</em> does not exist.</td>
</tr>
<tr>
<td>EPERM</td>
<td>The call is privileged and the caller was not the super-user.</td>
</tr>
<tr>
<td>ESRCH</td>
<td>No disk quota is found for the indicated user. Quotas have not been turned on for this file system.</td>
</tr>
<tr>
<td>EUSERS</td>
<td>The quota table is full.</td>
</tr>
<tr>
<td>EACCES</td>
<td>The quota file pointed to by <em>addr</em> exists but is not a regular file. The quota file pointed to by <em>addr</em> exists but is not on the file system pointed to by <em>special</em>.</td>
</tr>
<tr>
<td>EIO</td>
<td>Internal I/O error while attempting to read the quotas file pointed to by <em>addr</em>.</td>
</tr>
</tbody>
</table>

**FILES**  
`/usr/include/sys/fs/ufs_quota.h`  
`quotactl.h`  
Contains structure/function definitions and defines

**SEE ALSO**  
`quota(1M)`, `quotacheck(1M)`, `quotaon(1M)`, `getrlimit(2)`, `mount(2)`

Last modified 14 Sep 1995  
SunOS 5.7
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

rns_smt – Rockwell Station Management driver

SYNOPSIS

/dev/rns_smt

DESCRIPTION

On the Rockwell FDDI adapter boards, the rns_smt driver implements the FDDI Station Management protocol (SMT). The Station Management protocol includes Connection Management, Ring Management and all frame services. The rns_smt driver is a loadable, clonable STREAMS driver that can support multiple instances of the FDDI interface, as well as multiple application layer clients.

The cloning character-oriented devices /dev/rns_smt are used to access the rns_smt driver that supports Rockwell FDDI adapters. The /dev/rns_smt device is an interface used only for Station Management applications, such as those that gather MIB statistics or other Station information.

The SMT driver supports DLPI and SPI interfaces. All M_PROTO and M_PCPROTO type messages are interpreted as DLPI or SPI. SPI (SMT provider interface) is a Rockwell proprietary interface that is used during communication between the SMT and related applications. rns_smt is a "style 2" data link service provider, which means that an explicit DL_ATTACH_REQ is required to associate the opened stream with a particular device or physical point of attachment (PPA).

FILES

/dev/rns_smt interface used for Station Management applications

/kernel/drv/rns_smt.conf configuration file

Last modified 10 Apr 1996 SunOS 5.7 368
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, AF_INET);

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 may
only be carried out by the super user.

The operating system may 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 may be
achieved by using a mask of all zeros, and there may 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 (i.e., 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.

Note: 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 shown in the SYNOPSIS section above. 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 may use an additional sequence field to distinguish between outstanding messages. However, message replies may be lost when kernel buffers are exhausted.

The kernel may reject certain messages, and will indicate this by filling in the rtm_errno field of the rt_msghdr struct (see below). The following codes may be returned:

- **EEXIST** If requested to duplicate an existing entry
- **ESRCH** If requested to delete a non-existent entry
- **ENOBUS** If insufficient resources were available to install a new route.

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 may avoid the expense of reading replies to its own messages by issuing a `setsockopt(3N)` call indicating that the SO_USELOOPBACK option at the SOL_SOCKET level is to be turned off. A process may ignore all messages from the routing socket by doing a `shutdown(3N)` system call for further input.

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.

### Messages

User processes can obtain information about the routing entry to a specific destination by using a RTM_GET message.

Messages include:

```c
#define RTM_ADD /* Add Route */
#define RTM_DELETE /* Delete Route */
```
#define RTM_CHANGE /* Change Metrics, Flags, or Gateway */
#define RTM_GET /* Report Information */
#define RTM_LOOSING /* Kernel Suspects Partitioning */
#define RTM_REDIRECT /* Told to use different route */
#define RTM_MISS /* Lookup failed on this address */
#define RTM_RESOLVE /* request to resolve dst to LL addr */
#define RTM_NEWADDR /* address being added to iface */
#define RTM_DELADDR /* address being removed from iface */
#define RTM_IFINFO /* iface going up/down etc. */

A message header consists of:

```c
struct rt_msghdr {
    ushort_t rtm_msghlen; /* 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 sockaddr 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 */
    ulong_t rtm_inits; /* which values we are initializing */
    struct rt_metrics rtm_rmx; /* metrics themselves */
};
```

where,

```c
struct rt_metrics {
    ulong_t rmx_locks; /* Kernel must leave these values alone */
    ulong_t rmx_mtu; /* MTU for this path */
    ulong_t rmx_hopcount; /* max hops expected */
};
```
ulong_t rmx_expire; /* lifetime for route, e.g., redirect */
ulong_t rmx_recvpipe; /* inbound delay-bandwidth product */
ulong_t rmx_sendpipe; /* outbound delay-bandwidth product */
ulong_t rmx_ssthresh; /* outbound gateway buffer limit */
ulong_t rmx_rtt; /* estimated round trip time */
ulong_t rmx_rttvar; /* estimated rtt variance */
ulong_t rmx_pkcnt; /* packets sent using this route */
);

Flags include the values:

#define RTF_UP /* route usable */
#define RTF_GATEWAY /* destination is a gateway */
#define RTF_HOST /* host entry (net otherwise) */
#define RTF_REJECT /* host or net unreachable */
#define RTF_DYNAMIC /* created dynamically (by redirect) */
#define RTF_MODIFIED /* modified dynamically (by redirect) */
#define RTF_DONE /* message confirmed */
#define RTF_MASK /* subnet mask present */
#define RTF_CLONING /* generate new routes on use */
#define RTF_XRESOLVE /* external daemon resolves name */
#define RTF_LLINFO /* generated by ARP */
#define RTF_STATIC /* manually added */
route(7P) Protocols

```plaintext
#define RTF_BLACKHOLE /* just discard pkts (during updates) */
#define RTF_PROTO1 /* protocol specific routing flag #1 */
#define RTF_PROTO2 /* protocol specific routing flag #2 */
```

Specifiers for metric values in rmx_locks and rtm_inits are:

```plaintext
#define RTV_MTU /* init or lock _mtu */
#define RTV_HOPCOUNT /* init or lock _hopcount */
#define RTV_RPIPE /* init or lock _recvpipe */
#define RTV_SPIPE /* init or lock _sendpipe */
#define RTV_SSTHRESH /* init or lock _ssthresh */
#define RTV_RTT /* init or lock _rtt */
#define RTV_RTTVAR /* init or lock _rttvar */
```

Specifiers for which addresses are present in the messages are:

```plaintext
#define RTA_DST /* destination sockaddr present */
#define RTA_GATEWAY /* gateway sockaddr present */
#define RTA_NETMASK /* netmask sockaddr present */
#define RTA_GENMASK /* cloning mask sockaddr present */
#define RTA_IFP /* interface name sockaddr present */
#define RTA_IFA /* interface addr sockaddr present */
```


```
#define RTA_AUTHOR /* sockaddr for author of redirect */
#define RTA_BRD /* for NEWADDR, broadcast or p-p dest addr */
```

**SEE ALSO**

`ioctl(2), setsockopt(3N), shutdown(3N), routing(7P)`

**NOTES**

Some of the metrics may not be implemented and return zero. The implemented metrics are set in `rtm_inits`. 
NAME

routing – system support for packet network routing

DESCRIPTION

The network facilities provide general packet routing. 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 \texttt{ioctl(2)} commands, \texttt{SIOCADDRT} and \texttt{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 \texttt{/usr/include/net/route.h}:

\begin{verbatim}
struct rtenry {
    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 */
}
\end{verbatim}

with \texttt{rt\_flags} defined from:

\begin{verbatim}
#define RTF_UP 0x1 /* route usable */
#define RTF_GATEWAY 0x2 /* destination is a gateway */
#define RTF_HOST 0x4 /* host entry (net otherwise) */
\end{verbatim}

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
install 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 \texttt{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 \texttt{/dev/ip} device.

The \texttt{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.

- **ENOBUFFS**
  
  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

- \texttt{/dev/ip}
  
  IP device driver

### SEE ALSO

- \texttt{route(1M)}, \texttt{ioctl(2)}
NAME
sad – STREAMS Administrative Driver

SYNOPSIS
#include <sys/types.h>

#include <sys/conf.h>

#include <sys/sad.h>

#include <sys/stropts.h>

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 sap_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.
EINV\textsc{al}\ The major device number is invalid, the number of modules is invalid, or the list of module names is invalid.

ENO\textsc{str}\ The major device number does not represent a STREAMS driver.

EE\textsc{exist}\ The major-minor device pair is already configured.

ER\textsc{ange}\ The command is SAP\_\textsc{range} and sap\_lastminor is not greater than sap\_minor, or the command is SAP\_\textsc{clear} and sap\_minor is not equal to the first minor in the range.

ENO\textsc{dev}\ The command is SAP\_\textsc{clear} and the device is not configured for autopush.

ENOSR An internal autopush data structure cannot be allocated.

SAD\_\textsc{gap}\ Allows any user to query the sad driver to get the autopush configuration information for a given device. \textit{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, \textit{errno} is set to one of the following values:

EFAULT \textit{arg} points outside the allocated address space.

EINV\textsc{al}\ The major device number is invalid.

ENO\textsc{str}\ The major device number does not represent a STREAMS driver.

ENODEV\ The device is not configured for autopush.

SAD\_\textsc{vml}\ Allows any user to validate a list of modules (that is, to see if they are installed on the system). \textit{arg} is a pointer to a str\_\textsc{list} structure with the following members:

\begin{verbatim}
int sl_nmods;
struct str_mlist *sl_modlist;
\end{verbatim}

The str\_mlist structure has the following member:
char l_name[FNAMESZ+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
sbpro – Sound Blaster Pro, Sound Blaster 16, and Sound Blaster AWE32 audio device driver

SYNOPSIS
sbpro:sound, sbpro sbpro:sound, sbpro:ctl

DESCRIPTION
The Creative Labs Sound Blaster family of audio cards comprises DMA-capable ISA bus plug-in cards that provide 8 and 16 bit mono and stereo digitized sound recording and playback over a wide range of sampling rates. Each card includes a digital sound processor and mixing capability. Some of the cards also support more advanced audio features such as FM synthesis, advanced signal processing, advanced wave effects, and MIDI capability; however, the sbpro driver does not currently support those advanced features. The features and interfaces supported by the Solaris sbpro driver are described here and in audio(7I).

Some Sound Blaster cards support optional non-audio capabilities such as SCSI interfaces and CD-ROM interfaces. These interfaces are not supported by the sbpro driver. The Sound Blaster 16 optional SCSI-2 interface is supported by the aic(7D) driver.

The sbpro driver also supports certain "Sound Blaster compatible" audio devices, including some based on the ESS688 audio chip.

In addition, the driver supports some devices based on the Analog Devices AD1847 and AD1848, and Crystal Semiconductor CS4231 chips. Any CS4231-based devices supported by this driver are programmed in AD1848 compatibility mode. There is no special support in this driver for the more advanced CS4231 features. This family of devices will be referred to as the "AD184x family."

For a list of supported hardware implementations known to work with this driver, consult the latest version of the and/or the Solaris x86 Driver Update Guide (available online on the World Wide Web and other locations). The guide will contain more specific information about the settings for each type of card or motherboard.

APPLICATION PROGRAMMING INTERFACE

The Sound Blaster device is treated as an exclusive resource: only one process may open the device at a time. Since the Sound Blaster hardware does not support simultaneous sound input and output, the sbpro driver does not allow the simultaneous access of the device by two processes, even if one tries to open it read-only and the other write-only.

The sbpro driver will return "SUNW, sbpro" or "SUNW, sb16" in the name field of the audio_device structure. The version field will contain the version number of the card’s DSP chip, and the config field will be set to "SBPRO" or "SB16". The AWE32 is currently identified as an SB16. In all the discussion below, the Sound Blaster AWE32 behaves the same as the Sound Blaster 16.
Audio Data Formats

The Sound Blaster Pro handles 8-bit samples. In mono mode, audio data may be sampled at rates from 4,000 to 44,100 samples per second. In stereo mode, samples may be handled at the rates of 11,025 and 22,050 samples per second. The SB-16 can sample 8-bit or 16-bit mono or stereo data in the range of 5,000 to 44,100 Hz. Devices in the AD184x family can handle sample rates up to 48,000 Hz.

The Sound Blaster Pro hardware handles 8-bit linear samples in excess-128 format. The Sound Blaster 16 handles that format as well as 16-bit linear samples in two’s complement format. The sbpro driver will generate and accept data in these formats if AUDIO_ENCODING_LINEAR is selected in the encoding field of the audio information structure. 16 bit precision is not available on the Sound Blaster Pro. The sbpro driver will also accept and generate mu-law format data (as in the Greek letter mu) if the encoding field is set to AUDIO_ENCODING_ULAW. In this case, driver software performs the translation between linear and mu-law formats. mu-law encoding is designed to provide an improved signal-to-noise ratio at low amplitude levels. To achieve best results when using mu-law encoding, the audio record volume should be set so that typical amplitude levels lie within approximately three-fourths of the full dynamic range. Devices in the AD184x family support both mu-law and A-law in hardware, and the driver allows either of those encodings to be selected.

Audio Ports

The Sound Blaster hardware does not support multiple output devices, so the play.port field of the audio information structure only supports AUDIO_HEADPHONE. Output volume is controlled by software. There is a volume control thumbwheel on the back of the card which should be turned all the way up to maximum; otherwise no sound may be audible.

The record.port field of the audio information structure allows selection of which audio source is used for recording, and may be set to one of AUDIO_MICROPHONE, AUDIO_LINE_IN, or AUDIO_CD. These select input from the microphone jack, line-level input jack, or internal CD input, respectively. The microphone input is treated as a mono source by the hardware, although the microphone jack is a stereo jack. If your microphone has a mono plug, you should convert it to a stereo plug using an appropriate adapter. Line and CD are stereo sources. When recording in mono mode, both stereo channels are mixed before recording.

FILES

```
/dev/audio       linked to /dev/sound/0
/dev/audioctl    linked to /dev/sound/0ctl
/dev/sound/0     first audio device in the system
/dev/sound/0ctl  audio control for first audio device
```

Last modified 1 Jan 1997
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO

audioconvert(1), ioctl(2), attributes(5), aic(7D), audio(7I), streamio(7I)


BUGS

The current driver implementation does not support the A-law encoding mode for Sound Blaster and compatible devices.

The conversion of mu-law to 8-bit linear format for Sound Blaster and compatible devices can cause a loss of precision, resulting in poor sound quality in cases where the original recording level was well below normal. If this occurs while using the Sound Blaster 16 card, audioconvert(1) can be used to convert the original mu-law data to 16-bit linear format before play. This will preserve all the precision from the original mu-law sample.
NAME
sd – driver for SCSI disk and (ATAPI/SCSI) CD-ROM devices

SYNOPSIS
sd@target, lun:partition

DESCRIPTION

SPARC
This driver handles embedded SCSI-2 and CCS-compatible SCSI disk drives, CD-ROM drives, ATAPI 2.6 (SFF-8020i) compliant CD-ROM drives and the Emulex MD21 disk controller for ESDI drives. Note that support for the MD21 disk controller may be phased out after this release.

The type of disk drive is determined using the SCSI/ATAPI 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.

x86
This driver handles embedded SCSI-2 and CCS-compatible SCSI disk drives, CD-ROM drives, and ATAPI 2.6 (SFF-8020i) compliant CD-ROM drives.

x86 systems have a BIOS legacy that requires the presence of a master boot record (MBR) and fdisk table in the first physical sector of the bootable media. The Solaris disk label on an x86 hard disk is therefore always found in the second 512-byte sector within the fdisk partition if one is present.

Device Special Files
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; the names of the raw files 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 which do not meet the restrictions will cause the driver to return an EINVAL error. I/O requests to the block device have no alignment or length restrictions.

CD-ROM Drive Support
A CD-ROM disk is single-sided containing approximately 640 megabytes of data or 74 minutes of audio.

When the device is first opened, the CD-ROM drive’s eject button will be disabled, preventing the manual removal of the disk until the last close(2) is called.

No volume label is required on the CD-ROM. The disk geometry and partitioning information is always the same. If the CD-ROM is in ISO 9660 or High Sierra Disk format, it can be mounted as a file system.
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.

Refer to dkio(7I) and cdio(7I).

Permission denied.

The partition was opened exclusively by another thread.

The argument was a bad address.

Invalid argument.

An I/O error occurred.

This indicates that the device does not support the requested ioctl function.

During opening, the device did not exist. During close, the unlock of the drive failed.

The device is a read-only device.

The sd driver can be configured by defining properties in sd.conf file. Following are the supported properties:

- **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. Setting this value to 0 (zero) prevents the sd driver, in
its error-recovery process, from calling \texttt{scsi\_reset}(9F) with a second argument of "\ldots, \texttt{RESET\_TARGET}". This type of \texttt{scsi\_reset()} call will probably result in the HBA driver sending a SCSI Bus Device Reset message. The \texttt{scsi\_reset()} call with a second argument of "\ldots, \texttt{RESET\_TARGET}" may still result from an explicit request via the \texttt{USCSICMD} ioctl. Some high-availability multi-initiator systems may wish to prohibit the Bus Device Reset message; they can effectively do so by setting the \texttt{allow-bus-device-reset} property to 0.

\section*{FILES}

\begin{itemize}
  \item \texttt{sd.conf} \hspace{1cm} driver configuration file
  \item \texttt{/dev/dsk/cntndsn} \hspace{1cm} block files
  \item \texttt{/dev/rdsn/cntndsn} \hspace{1cm} raw files
\end{itemize}

where:

\begin{itemize}
  \item \texttt{cn} \hspace{1cm} controller \texttt{n}
  \item \texttt{tn} \hspace{1cm} SCSI target id \texttt{n} (0-6)
  \item \texttt{dn} \hspace{1cm} SCSI LUN \texttt{n} (0-7 normally; some HBAs support LUNs to 15 or 32, see the specific manpage for details)
  \item \texttt{sn} \hspace{1cm} partition \texttt{n} (0-7)
\end{itemize}

\section*{SEE ALSO}

\begin{itemize}
  \item \texttt{fdisk(1M)}, \texttt{format(1M)}, \texttt{close(2)}, \texttt{ioctl(2)}, \texttt{lseek(2)}, \texttt{read(2)}, \texttt{write(2)}, \texttt{kstat(3K)}, \texttt{driver.conf(4)}, \texttt{scsi(4)}, \texttt{cdio(7I)}, \texttt{dkio(7I)}, \texttt{scsi\_ifsetcap(9F)}, \texttt{scsi\_reset(9F)}
\end{itemize}

\texttt{ANSI Small Computer System Interface-2 (SCSI-2)}

\texttt{Emulex MD21 Disk Controller Programmer Reference Manual}

\texttt{ATA Packet Interface for CD-ROMs, SFF-8020i}

Last modified 29 Apr 1998 SunOS 5.7 386
DIAGNOSTICS

| Error for Command: `<command name>` | Error Level: Fatal |
| Requested Block: `<n>` | Error Block: `<m>` |
| Vendor: `<vendor name>` | 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 key.

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 was probably 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

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.
sdrestart transport failed (<n>)

Driver attempted to retry a command and experienced a transport error.

transfer length not modulo <n>

Illegal request size.

transport of request sense fails (<n>)

Driver attempted to submit a request sense command and failed.

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  
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, together with basic asynchronous and synchronous communication support. This manual page describes the asynchronous protocol interface, the synchronous interface is described under se_hdlc(7D). Both interfaces use the same driver, but there is little overlap in the format of the interfaces, so a separate manual seems appropriate. bus_address is the platform specific se device bus address. port_name is a single letter (a-z).

APPLICATION PROGRAMMING INTERFACE  
The Siemens 82532 provides two serial input/output channels that are capable of supporting a variety of communication protocols. A typical system will have one of these devices to implement two serial ports (port_name), usually configured for RS-423 (which will also support most RS-232 equipment). The 82532 uses 64 character input and output FIFOs to reduce system overhead. When receiving characters the CPU is notified when 32 characters have arrived (1/2 of receive buffer is full) or no character has arrived in the time it would take to receive 4 characters at the current baud rate. When sending characters the 82532 places the first 64 characters to be sent into its output FIFO and thereafter notifies the CPU when it is half empty (32 characters left). Delays may be seen when modifying the port’s attributes while the se driver waits for the 82532 to transmit the remaining characters within its output FIFO before making the requested change.

The chip implements CTS/RTS flow control in hardware. Removing the responsibility for this feature from the CPU prevents data overruns 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 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.
Devices 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.

/dev/cua/[a-z], /dev/term/[a-z] and /dev/tty[a-z] are valid name space entries. The number of entries used in this name space are machine dependent.

The /dev/tty[a-z] device names only exist if the Binary Compatibility Package is installed. The /dev/tty[a-z] device names are created by the ucblinks command. This command is only available with the Binary Compatibility Package.

To allow a single tty line to be connected to a modem and used for both incoming and outgoing calls, a special feature, controlled by the minor device number, is available. 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.

Once 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, after which Carrier Detect will usually drop as well) and carrier is detected again, and a non-blocking open will return an error. Also, 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, 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 also used for dial-out (by tip(1) or uucp(1C)) as /dev/cua/[a-z] when no one is logged in on the line.

IOCTLS

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

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, TIOCMBS, and TIOCMBS1 ioctl commands.

The input and output line speeds may be set to all baud rates supported by termio. The speeds cannot be set independently; when the output speed is set, the input speed is set to the same speed.
When using baud rates over 100,000 baud, the software will change the line
driver configuration to handle the higher data rates. This decreases the
theoretical maximum cable length from 70 meters to 30 meters. For further
details, see the Unitrode UC5170 line driver data sheets under "Slew rate
programming".

**ERRORS**

An open() will fail if:

- **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

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>SPARC</td>
</tr>
</tbody>
</table>

**SEE ALSO**

cu(1C), tip(1), ucblinks(1B), uucp(1C), ports(1M), attributes(5),
ioctl(2), open(2), ldterm(7M), se_hdlc(7D), termio(7I),
ttcompat(7M), zs(7D), zsh(7D)

*Binary Compatibility Guide*
The 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 too high a data rate.

The se driver was unable to store data it removed the 82532 FIFO. This suggests that 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); this will 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@
bu_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 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 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 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_hclc ioctl()` commands:

```c
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 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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>SPARC</td>
</tr>
</tbody>
</table>

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: 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_hdlc_start: Invalid message type d on write queue

driver received an invalid message type from streams.

se_hdlc: 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  

DESCRIPTION  

The ses device driver is an interface to SCSI enclosure services devices. These devices sense and monitor the physical conditions within 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  A sample output of ses.

Following is an example of the ses.conf file format:

```bash
# 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;
```

FILES  

/kernel/drv/ses.conf  driver configuration file

ATTRIBUTES  

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>SPARC</td>
</tr>
</tbody>
</table>

SEE ALSO  

ssaadm(1M), driver.conf(4), attributes(5), esp(7D), isp(7D), ioctl(9E)

SunOS 5.7  Last modified 22 May 1997
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:

```c
uint32_t size; /* Size of buffer which follows */
uint8_t page_code; /* Page to be read/written */
uint8_t reserved[3]; /* Reserved; Set to 0 */
unit8_t buffer[1]; /* Size arbitrary, user specifies */
```

EXAMPLES
EXAMPLE 1 Using the SES_IOCTL_GETSTATE ioctl.

Following is an example of using the SES_IOCTL_GETSTATE ioctl to recover
20 bytes of page 4 from a previously opened device.

```c
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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>SPARC</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>SPARC</td>
</tr>
</tbody>
</table>

SEE ALSO

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 \texttt{sf\%d}, where \%d is the instance number of the \texttt{sf} device.

\textbf{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.

\textbf{Bad soft state}

Driver requested an invalid internal state structure. Driver did not attach to device, SCSI devices will be inaccessible.

\textbf{Failed to obtain transport handle}

Driver was unable to obtain a transport handle to communicate with the \texttt{socal} driver. Driver did not attach to device, SCSI devices will be inaccessible.

\textbf{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.

\textbf{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.

\textbf{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.

\textbf{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.

\textbf{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.
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
smartii – Compaq Smart-2 EISA/PCI and Smart-2SL PCI Array Controller
driver

DESCRIPTION
The smartii driver is a driver for Compaq Smart-2 EISA/PCI and Smart-2SL
PCI Array Controllers on Compaq Servers. The driver supports magnetic fixed
disks and magnetic removable disks.

The Smart-2 and Smart-2SL controllers can be configured using the Compaq
Array configuration utility. Each Smart-2 controller can support a maximum of
14 physical disks and each Smart-2SL controller can support a maximum of 7
disks. Only one bus can be used at any time for the Smart-2SL controller. Each
controller can support 32 logical volumes.

The block files access the disk using the system’s normal buffering mechanism
and they 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; the names of the raw files are found in /dev/rdsk.

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 Solaris fdisk
partition. Other slices may be used for usr file systems or system reserved
areas.

fdisk partition 0 is to access the entire disk and is generally used by the
fdisk(1M) program.

FILES
/dev/dsk/cndn[s|p]n block device
/dev/rdsk/cndn[s|p]n raw device where:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cn</td>
<td>controller n</td>
</tr>
<tr>
<td>dn</td>
<td>lun n (0-7)</td>
</tr>
<tr>
<td>sn</td>
<td>UNIX system slice n</td>
</tr>
<tr>
<td></td>
<td>(0-15)</td>
</tr>
<tr>
<td>pn</td>
<td>fdisk partition (0)</td>
</tr>
</tbody>
</table>

ATTRIBUTES
See attributes(5) for descriptions of the following attributes:
### Devices

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

**SEE ALSO**  
smart2cfg(1), fdisk(1M), attributes(5), cmdk(7D), csa(7D)

Last modified 20 Jun 1997

SunOS 5.7

406
NAME

smc – SMC 8003/8013/8216/8416 Ethernet device driver

SYNOPSIS

#include <sys/stropts.h>
#include <sys/ethernet.h>
#include <sys/dlpi.h>
#include <sys/gld.h>

DESCRIPTION

The SMC 8003/8013/8216/8416 Ethernet driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface, dlpi(7P), over the SMC 80X3/8216/8416 Ethernet controllers. The smc driver is dependent on /kernel/misc/gld, a loadable kernel module that provides the smc driver with the DPLI and STREAMS functionality required of a LAN driver. See gld(7D) for more details about the primitives supported by the smc driver.

Multiple SMC controllers installed within the system are supported by the driver. The smc driver provides basic support for the SMC hardware. Functions include chip initialization, frame transmit and receive, multicast and "promiscuous" support, and error recovery and reporting.

APPLICATION PROGRAMMING INTERFACE

The cloning character-special device /dev/smc is used to access all SMC controllers installed within the 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 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 or DL_CSMACD.
- The broadcast address value is Ethernet/IEEE broadcast address (FF:FF:FF:FF:FF:FF).

CONFIGURATION

Older 8-bit SMC cards have a restricted set of allowable configurations under the Solaris system. See the Configuring Devices Module in the Information Library for more details.

FILES

/kernel/drv/smc character special device
/kernel/drv/smc.conf smc configuration file

407 SunOS 5.7 Last modified 14 May 1997
ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO attributes(5), dlpi(7P), gld(7D), streamio(7I)
NAME
smce – SMC 3032/EISA dual-channel Ethernet device driver

SYNOPSIS
#include <sys/stropts.h>
#include <sys/ethernet.h>
#include <sys/dlpi.h>

DESCRIPTION
The smce Ethernet driver is a multi-threaded, loadable, clonable, STREAMS
hardware driver supporting the connectionless Data Link Provider Interface,
dlpi(7P), over the SMC 3032/EISA dual-channel Ethernet controllers. Each
dual-channel 3032/EISA controller can support two subnetworks. Multiple
3032/EISA controllers installed within the system are supported by the driver.
The smce driver provides basic support for the 3032/EISA hardware.
Functions include chip initialization, frame transmit and receive, multicast and
“promiscuous” support, and error recovery and reporting on both channels.

APPLICATION
PROGRAMMING INTERFACE
The cloning, character-special device /dev/smce is used to access all 3032/
EISA devices installed within the system.

The smce driver is a “style 2” Data Link Service provider. All M_PROTO and
M_PCPROTO type messages 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 integer and indicates the corresponding device instance (unit)
number. The unit numbers are assigned sequentially to each board found. For
the dual-channel 3032/EISA controller, a pair of ppa IDs is associated with
each controller. The lower (even) numbered ppa corresponds to port A of the
controller while the higher (odd) numbered ppa corresponds to port B. The
search order is determined by the order defined in the
/kernel/drv/smce.conf file. If the ppa field value does not correspond to
a valid device instance number for this system, the driver will return an error
(DL_ERROR_ACK). The device is initialized on first attach and de-initialized
(stopped) on 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).
- The minimum SDU is 0. The driver will pad to the mandatory 60-octet
  minimum packet size.
- The dl 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.

- 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 (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 Pointer (SAP) with the stream. The smce 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.

In addition to Ethernet V2 service, an “802.3 mode” is also provided by the driver. In this mode, sap values in the range [0-1500] are treated as equivalent and represent a desire by the user for “802.3” mode. If the value of the sap field of the DL_BIND_REQ message is within this range, then the driver expects that the destination DLSAP in a DL_UNITDATA_REQ will contain the length of the data rather than a sap value. All frames received from the media that have a “type” field in this range are assumed to be 802.3 frames, and they are routed up all open streams which are bound to any sap value within this range. 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.

The smce 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 should instead 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 smce driver. The smce driver will route received Ethernet frames up all open and bound streams that have 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.

smce 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. When the system starts up, both channels on the 3032/EISA controller uses the same Ethernet address obtained from the on-board EEPROM.

The DL_PHYSET_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 an EPERM error 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 on this stream is successful. Once changed, all streams 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 system is rebooted, whichever comes first.

CONFIGURATION

The /kernel/drv/smce.conf file is shipped pre-configured to handle all supported card configurations. It is not intended to be modified in the field.
When configuring the card, it is important to ensure that there are no conflicts between the board’s I/O port or IRQ level, and those of any other device in the system.

FILES
/dev/smce smce cloning, character-special device
/kernel/drv/smce.conf smce configuration file.

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO attributes(5), dlpi(7P)
NAME
smceu – SMC Elite32 Ultra (8232) Ethernet device driver

SYNOPSIS
#include <sys/stropts.h>
#include <sys/stream.h>
#include <sys/dlpi.h>
#include <sys/ethernet.h>

DESCRIPTION
The smceu Ethernet driver is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface over an SMC Elite32 Ultra (8232) EISA-bus adapter. The smceu driver is dependent on /kernel/misc/gld, a loadable kernel module that provides the smceu driver with the DLPI and STREAMS functionality required of a LAN driver. See gld(7D) for more details about primitives supported by the smceu driver.

The smceu driver provides basic support for the SMC Elite32 Ultra hardware. Functions include chip initialization, frame transmission and reception, multicast and promiscuous support, and error recovery and reporting.

Multiple SMC Elite32 Ultra controllers installed within the system are supported by the driver.

APPLICATION
PROGRAMMING INTERFACE
The cloning character-special device /dev/smceu is used to access all SMC Elite32 Ultra controllers installed within the system.

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

- The maximum SDU is 1500 (ETHERMTU).
- The minimum SDU is 0. Note that the smceu driver will pad to the mandatory 60-octet minimum packet size.
- The dlsap address length is ETHERADDRL + 2.
- 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).
CONFIGURATION
The driver is shipped configured to support any legal device configuration. See the SMC Elite32 Ultra (8232) Device Configuration page in the for information regarding hardware configuration.

FILES
/dev/smceu character special device
/kernel/drv/smceu smceu driver
/kernel/drv/smceu.conf configuration file

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO
attributes(5), dlpi(7P), gld(7D)
NAME
smcf – SMC Ether100 (9232) Ethernet device driver

SYNOPSIS
#include <sys/stropts.h>
#include <sys/stream.h>
#include <sys/dlpi.h>
#include <sys/ethernet.h>

DESCRIPTION
The smcf Ethernet driver is a multi-threaded, loadable, clonable, STREAMS
hardware driver supporting the connectionless Data Link Provider Interface
over an SMC Ether100 (9232) Fast Ethernet EISA-bus adapter. The smcf driver
is dependent on /kernel/misc/gld, a loadable kernel module that provides
the smcf driver with the DLPI and STREAMS functionality required of a LAN
driver. See gld(7D) for more details about primitives supported by the smcf
driver.

The smcf driver provides basic support for the SMC Ether100 hardware.
Functions include chip initialization, frame transmission and reception,
multicast and promiscuous support, and error recovery and reporting.

Multiple SMC Ether100 controllers installed within the system are supported
by the driver.

APPLICATION
PROGRAMMING
INTERFACE
The cloning character-special device /dev/smcf is used to access all SMC
Elite32 Ultra controllers installed within the system.

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

- The maximum SDU is 1500 (ETHERMTU).
- The minimum SDU is 0. Note that the smcf driver will pad to the
  mandatory 60-octet minimum packet size.
- The dlsap address length is ETHERADDRL + 2.
  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). The smcf driver can support up to 16 multicast
  addresses on each adapter.

CONFIGURATION
The driver is shipped configured to support any legal device configuration. See
the SMC Ether100 (9232) Device Configuration page in the for information
regarding hardware configuration.
FILES
/dev/smcf character special device
/kernel/drv/smcf smcf driver
/kernel/drv/smcf.conf configuration file

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO
attributes(5), dlpi(7P), gld(7D)
NAME  soc – Serial Optical Controller (SOC) device driver

SYNOPSIS  soc@<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 supports a point to point interface to another Fibre Channel device.

The soc device driver is a nexus driver. The soc driver implements portions of the FC-2 and FC-4 layers of the Fibre Channel.

FILES  /kernel/drv/soc

ELF Kernel Module

SEE ALSO  sbus(4), pln(7D), ssd(7D)

Writing Device Drivers

DIAGNOSTICS  The messages described below are some that may appear on system console, as well as being logged.

On the console these messages are preceded by
"soc%d: port %a"

where %d is the instance number of the soc controller and %a is the port on the host adapter.

Fibre Channel is ONLINE

The Fibre Channel is now online to the device.

Fibre Channel is OFFLINE

The Fibre Channel connection is now offline.

INCORRECT WWN: Found: xxxx,xxxxxxxx Expected: yyyy,yyyyyyyy

This message means that the soc re-logged into a device after the Fibre Channel connection went offline and back online and the World Wide Name of the device is now different. This probably means the cable has been plugged into another device.

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: could not alloc offline packet structure

Driver was unable to allocate space for the internal state structure. Driver did not attach to device, devices will be inaccessible.
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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>SPARC</td>
</tr>
</tbody>
</table>

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 are some that may appear on system console, as well as 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(3N)) 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(3N)
**NAME**
ssd – driver for SPARCstorage Array and Fibre Channel Arbitrated Loop disk devices

**SYNOPSIS**
ssd@port, target:partition

**DESCRIPTION**
This driver handles both SCSI-2 disks in the SPARCstorage Array and Fibre Channel Arbitrated Loop (FC-AL) disks on Private loops.

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. 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; 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, partition 1 as a paging area (for example, swap), and partition 2 for backing 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.

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**
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.
The device does not support the requested ioctl function.

When returned during `open(2)`, this error indicates the device does not exist.

The device is a read-only device.

---

**FILES**

- `ssd.conf`: driver configuration file
- `/dev/dsk/cntndsn`: block files
- `/dev/rdsk/cntndsn`: raw files

where, for the SPARC storage Array:

- `cn`: is the controller number on the system. Each SPARC storage Array will have a unique controller number.
- `tn`: port number within the SPARC storage Array. 
- `dn`: SCSI target number.
- `sn`: partition number.

and for all FC-AL disks:

- `cn`: is the controller number on the system.
- `tn`: 7-bit disk loop identifier, such as switch setting.
- `dn`: SCSI lun number.
- `sn`: partition number (0-7).

---

**SEE ALSO**

- `format(1M)`, `ioctl(2)`, `lseek(2)`, `open(2)`, `read(2)`, `write(2)`, `driver.conf(4)`, `cdio(7I)`, `dkio(7I)`

---

**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>

Last modified 27 May 1997
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
  , u] [b] [n]

DESCRIPTION  
The st device driver is an interface to various SCSI tape devices. Supported tape devices include 1/4" Tandberg 2.5 Gigabyte QIC tape drive, 1/4" Archive Viper QIC-150 streaming tape drive, 1/4" Emulex MT-02 tape controller, HP-88780 1/2" tape drive, Exabyte EXB-8200/8500/8505/8505XL 8mm cartridge tape, and the Archive Python 4 mm DAT tape subsystem. st provides a standard interface to these various devices; see mtio(7I) for details.

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. Often tape format is also referred to as tape density.

The driver now reserves the tape drive upon open and releases it at close for use in multi-initiator environments. 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 the tape drive is opened in O_NDELAY mode, no reservation will occur during the open, as per the POSIX standard (see standards(5)). However, before the first tape operation or I/O occurs, a reservation will occur to provide reserve/release functionality.

Persistent Errors and Asynchronous Tape Operation

Read Operation

The st driver now supports persistent errors (see mtio(7I)) and asynchronous tape operations (see mtio(7I), aioread(3), and aiowrite(3)).

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 will detect a read failure on the first read and automatically switch 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 user-selected format. Other formats issue a warning message to inform the user of an overridden format selection. Some devices automatically perform this function and do not require driver support (1/2" reel tape drive, for example).

Last modified 20 Feb 1998  SunOS 5.7  428
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 tape driver has a built-in configuration table for all Sun supported tape drives. In order to support the addition of third party tape devices or to override a built-in 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. 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>;
```

Note that 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. 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:
<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:

```
<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_VARIABLE</td>
<td>0x0001</td>
</tr>
<tr>
<td>ST_QIC</td>
<td>0x0002</td>
</tr>
<tr>
<td>ST_REEL</td>
<td>0x0004</td>
</tr>
<tr>
<td>ST_BSF</td>
<td>0x0008</td>
</tr>
<tr>
<td>ST_BSR</td>
<td>0x0010</td>
</tr>
<tr>
<td>ST_LONG_ERASE</td>
<td>0x0020</td>
</tr>
<tr>
<td>ST_AUTODEN_OVERRIDE</td>
<td>0x0040</td>
</tr>
<tr>
<td>ST_NOBUF</td>
<td>0x0080</td>
</tr>
<tr>
<td>ST_KNOWS_EOD</td>
<td>0x0200</td>
</tr>
<tr>
<td>ST_UNLOADABLE</td>
<td>0x0400</td>
</tr>
<tr>
<td>ST_SOFT_ERROR_REPORTING</td>
<td>0x0800</td>
</tr>
<tr>
<td>ST_LONG_TIMEOUTS</td>
<td>0x1000</td>
</tr>
<tr>
<td>ST_BUFFERED_WRITES</td>
<td>0x4000</td>
</tr>
<tr>
<td>ST_NO_RECSIZE_LIMIT</td>
<td>0x8000</td>
</tr>
<tr>
<td>ST_MODE_SEL_COMP</td>
<td>0x10000</td>
</tr>
<tr>
<td>ST_NO_RESERVE_RELEASE</td>
<td>0x20000</td>
</tr>
<tr>
<td>ST_READ_IGNORE_ILI</td>
<td>0x40000</td>
</tr>
<tr>
<td>ST_READ_IGNORE_EOFS</td>
<td>0x80000</td>
</tr>
<tr>
<td>ST_SHORT_FILEMARKS</td>
<td>0x100000</td>
</tr>
</tbody>
</table>
```

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.
<table>
<thead>
<tr>
<th>ST_KNOWS_EOD</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_UNLOADABLE</td>
<td>The flag indicates the device will not complain if the st driver is unloaded and loaded again (see <code>modload(1M)</code> and <code>modunload(1M)</code>). That is, the driver will return the correct inquiry string.</td>
</tr>
<tr>
<td>ST_SOFT_ERROR_REPORTING</td>
<td>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.</td>
</tr>
<tr>
<td>ST_LONG_TIMEOUTS</td>
<td>The flag indicates the tape device requires timeouts that are 5 times longer than usual for normal operation.</td>
</tr>
<tr>
<td>ST_BUFFERED_WRITES</td>
<td>If the flag is set, when data is written to the tape device, the data is buffered by the driver. The application may receive acknowledgement of completion of the write request before the data has been written to tape.</td>
</tr>
<tr>
<td>ST_NO_RECSIZE_LIMIT (SPARC Only)</td>
<td>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</td>
</tr>
</tbody>
</table>
transfer size of the system. (Refer to Large Record Sizes and WARNINGS.)

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 will be enabled only if the "c" or "u" device is used. For any other device densities, compression will be 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 (suppress 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 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.

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 will be 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.

<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).

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 retention 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 A sample program.

Example of a global tape-config-list property:

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=0 lun=0;
name="st" class="scsi" target=1 lun=0;
name="st" class="scsi" target=2 lun=0;

Last modified 20 Feb 1998
Example of a tape-config-list property applicable to target 2 only:

```
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;
```

Large Record Sizes

**SPARC ONLY**

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.

Write Data Buffering

Tape drives with data compression require a much higher data rate in order to stream the tape. Write data buffering in the driver improves streaming to the drive without changing the application and augments the buffering in the tape drive itself. If write data buffering is enabled, data is buffered in the driver and the request is immediately acknowledged by the driver before it has been written to the tape drive. This enables the driver to submit the next request as soon as the previous request completes and the application to prepare the next request while the current request is in progress. A SCSI tape drive that allows buffering requires ORing the flag ST_BUFFERED_WRITES with other flags in the ‘options’ field in st.conf. (Refer to Tape Configuration.) By default, this option is set for the built-in config entries of the Archive DAT and Exabyte drives.

In order for write buffering to work properly, sufficient space after LEOT must be available to empty the write buffers. Older tape devices usually do not have sufficient space after LEOT.

To turn on tape buffering, a property in st.conf called "tape-driver-buffering" should be added. The value assigned to this property is the maximum number of buffered write requests allowed. For example, 0 indicates no write request buffering allowed, while 2 indicates buffer up to 2 write requests. If this property is not specified in st.conf, the driver defaults to a value of 0. The maximum size of write request that can be buffered is specified through a property in st.conf called "tape-driver-buf-max-size". If this property is not specified in st.conf, the driver defaults the buffer size to a value of 1 Mbyte.

An example of st.conf, where the maximum number of write requests buffered is 4 and maximum size of write request buffered is 2 Mbyte, is given below. This applies to all nodes in this conf file.

```
tape-driver-buffering = 4; tape-driver-buf-max-size = 0x200000; name="st" class="archive"
```

In the case of a SCSI bus reset, a medium error, or any other fatal transport error on a buffered request, the driver returns an error on subsequent write
requests and allows no more writes. If no further write requests occur, an error is returned on close.

Since some applications may perceive write buffering as a potential data integrity problem, this feature is disabled by default and needs to be explicitly enabled in the config entry and turned on by means of the property in \texttt{st.conf}. Furthermore, some fault tolerant backup servers make assumptions about the data buffering in the tape drive itself. These assumptions may not be valid if write buffering has been enabled.

Write buffering may be superseded by other performance enhancements in a future release.

\begin{files}
\texttt{/kernel/drv/st.conf}\\
\hspace{1em}driver configuration file\\
\texttt{/usr/include/sys/mtio.h}\\
\hspace{1em}structures and definitions for mag tape io control commands\\
\texttt{/usr/include/sys/scsi/targets/stdef.h}\\
\hspace{1em}definitions for SCSI tape drives\\
\texttt{/dev/rmt/[0-127][l,m,h,u,c][b][n]}\\
\end{files}

where \( l, m, h, u, c \) specifies the density (low, medium, high, ultra/compressed), \( b \) the optional BSD behavior (see \texttt{mtio(7)}), and \( n \) the optional no rewind behavior. For example, \texttt{/dev/rmt/0lbn} specifies unit 0, low density, BSD behavior, and no rewind.

For 1/2" reel tape devices (HP-88780), the densities are:

\begin{tabular}{|c|}
\hline
\texttt{l} & 800 BPI density \\
\texttt{m} & 1600 BPI density \\
\texttt{h} & 6250 BPI density \\
\texttt{c} & data compression \\
\texttt{(not supported on all modules)} \\
\hline
\end{tabular}

For 8mm tape devices (Exabyte 8200/8500/8505):
For 4mm DAT tape devices (Archive Python):

| l  | Standard format                      |
| m  | 5 Gbyte format (8500, 8505 only)     |
| h,c| 5 Gbyte compressed format (8505 only)|

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(3), aiowrite(3), kstat(3K), driver.conf(4), scsi(4), standards(5), esp(7D), isp(7D), mlio(7I), ioctl(9E)

DIAGNOSTICS

Error for command '<command name>' Error Level: Fatal
Requested Block <n>, Error Block: <m>
Sense Key: <sense key name>
Vendor '<name>': ASC = 0x<a> (<extended sense code 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, ASCQ and FRU information is returned by the target in response to a request sense command.

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 with device and bus reset. This recovery failed.

Periodic head cleaning required The driver reported that periodic head cleaning is now 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 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.

**WARNINGS**

In 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 Computer Corporation.

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

stc – Serial Parallel Communications driver for SBus

DESCRIPTION

The SPC/S SBus communications board consists of eight asynchronous serial ports and one IBM PS/2-compatible parallel port. The stc driver supports up to 8 SPC/S boards in an SBus system. Each serial port has full modem control: the CD, DTR, DSR, RTS and CTS modem control lines are provided, plus flow control is supported in hardware for either RTS/CTS hardware flow control or DC1/DC3 software flow control. The parallel port is unidirectional with support for the ACK, STROBE, BUSY, PAPER OUT, SELECT and ERROR interface signals. Both the serial and parallel ports support those termios(7I) device control functions specified by flags in the c_cflag word of the termios(3) structure; in addition, the serial ports support the IGNPAR, FARMK, INPCK, IXON, IXANY and IXOFF flags in the c_iflag word of the termios(3) structure. The latter c_iflag functions are performed by the stc driver for the serial ports. Since the parallel port is a unidirectional, output-only port, no input termios(3) (c_iflag) parameters apply to it. Trying to execute a nonsensical ioctl() on the parallel port is not recommended. All other termios(3) functions are performed by STREAMS modules pushed atop the driver. When an stc device is opened, the ldterm(7M) and ttcompat(7M) STREAMS modules are automatically pushed on top of the stream if they are specified in the /etc/iu.ap file (the default condition), providing the standard termio(7I) interface.

The device names of the form /dev/term/n or /dev/ttyyn specify the serial I/O ports provided on the SPC/S board, conventionally as incoming lines. The device names of the form /dev/cua/n or /dev/ttyzn specify the serial I/O ports provided on the SPC/S board, conventionally as outgoing lines. The device names of the form /dev/printers/n or /dev/stclpn specify the parallel port, and the device name of the form /dev/stcn specify a special control port per board.

To allow a single tty line to be connected to a modem and used for both incoming and outgoing calls, a special feature, controlled by the minor device number, has been added. Minor device numbers in the range 128-191 correspond to the same physical lines as those in the range 0-63 (that is, the same line as the minor device number minus 128).

A dial-in line has a minor device in the range 0-63 and is conventionally named /dev/term/n, where n is a number indicating which dial-in line it is (so that /dev/term/0 is the first dial-in line), and the dial-out line corresponding to that dial-in line has a minor device number 128 greater than the minor device number of the dial-in line and is conventionally named /dev/cua/n, where n is the number of the dial-in line. These devices will also have the compatibility names /dev/ttyzn.

The /dev/cua/n lines are special in that they can be opened even when there is no carrier on the line. Once a /dev/cua/n line is opened, the corresponding
/dev/term/* line cannot be opened until the /dev/cua/* line is closed; a
blocking open will wait until the /dev/cua/* line is closed (which will drop
DTR, after which DCD will usually drop as well) and carrier is detected again,
and a non-blocking open will return an error. Also, if the /dev/term/* line
has been opened successfully (usually only when carrier is recognized on the
modem) the corresponding /dev/cua/* line can not be opened. This allows a
modem to be attached to /dev/term/0, for example, and used for dial-in (by
enabling the line for login (using pmadm(1M)) and also used for dial-out (by
tip(1) or uucp(1C)) as /dev/cua/0 when no one is logged in on the line.

The parallel port is given the name /dev/stclp, where n is the SPC/S unit
number (see Minor Numbers, below).

The control port, named /dev/stcn, where n is the SPC/S, is available. And
ioctl() is provided for this special file which allow the collection of statistics
maintained on serial port performance.

Minor Numbers

<table>
<thead>
<tr>
<th>o</th>
<th>p</th>
<th>u</th>
<th>u</th>
<th>lll</th>
</tr>
</thead>
<tbody>
<tr>
<td>o</td>
<td>set if this device is an outgoing serial line</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>set if this device is a parallel port device</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>u</td>
<td>device unit number</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| l  | device line number if this is the parallel port line, 'p' should be 1 and
    |     'lll' should be all 0's if this is the control line, both 'p' and 'lll'
    |     should be set to all 1's |

IOCTLS

The standard set of termio ioctl() calls are supported by the stc driver on
both the serial and parallel ports.

If the CRTSCTS flag in the c_cflag is set and if CTS is high, output will be
transmitted; if CTS is low, output will be frozen. If the CRTSCTS flag is clear,
the state of CTS has no effect. Breaks can be generated by the TCSBRK,
TIOCSBRK and TIOCCBRK ioctl() calls. The modem control lines TIOCM_CAR,
TIOCM_CTS, TIOCM_RTS, TIOCM_DSR and TIOCM_DTR are provided for the
serial ports, although the TIOCMGET ioctl() call will not return the state of the
TIOCM_RTS or TIOCM_DSR lines, which are output-only signals.

The serial port input and output line speeds may be set to any of the speeds
supported by termio(7I).

DEVICE-SPECIFIC

IOCTLS

The following additional ioctl()’s are supported by the stc driver.

STC_SPPC( struct ppc_params_t *) set parallel port parameters
(valid until changed or close())
The possible values for flags, defined in /usr/include/sys/stcio.h, are:

- **PP_PAPER_OUT**: honor PAPER OUT from port; returned HIGH means PAPER OUT.
- **PP_ERROR**: honor ERROR from port; returned HIGH means ERROR.
- **PP_BUSY**: honor BUSY from port; returned HIGH means BUSY.
- **PP_SELECT**: honor SELECT from port; returned HIGH means OFFLINE.
- **PP_MSG**: print console message on every error scan.
- **PP_SIGNAL**: send a PP_SIGTYPE (SIGURG) to the process if printer error.

The **state** field contains the current status of the printer interface. It is analogous to the bit order of **flags**, but contains the status the driver maintains, masked by the flags that are set. The result of shifting **state** **PP_SHIFT** bits to the left is the actual state of the hardware.

The **STC_SPPC** and **STC_GPPC** ioctl calls are understood only by the parallel port.

The STC_GSTATS ioctl calls are understood only by the parallel port.
struct stcb fellows

uint_t set_params; /* call to stc_param */
uint_t no_start; /* can’t run in stc_start(); already there */
uint_t xmit_int; /* transmit interrupts */
uint_t rcv_int; /* receive interrupts */
uint_t rcvex_int; /* receive exception interrupts */
uint_t modem_int; /* modem change interrupts */
uint_t xmit_cc; /* characters transmitted */
uint_t rcv_cc; /* characters received */
uint_t break_cnt; /* BREAKs received */
uint_t bufcall; /* times we couldn’t get STREAMS buffer */
uint_t canwait; /* stc_drainsilo() called w/pending timer */
uint_t reserved; /* this field is meaningless */
};

The possible cmd values, defined in /usr/include/sys/stcio.h, are

STAT_CLEAR clear the line statistics
STAT_GET get the line statistics

The stc_GSTATS ioctl works only on the SPC/S control port.

Several methods may be used to enable or disable soft carrier on a particular
serial line. The non-programmatic method is to edit the
/platform/platform/kernel/drv/stc.conf file. For this change to take
effect, the machine must be rebooted. See the next section,
SETTING DEFAULT LINE PARAMETERS, for more information on this
method. From within an application program, you can enable or disable the
recognition of carrier on a particular line by issuing the TIOCGSOFTCAR ioctl()
to the driver.

The default mode of operation for the DTR signal is to assert it on the first
open() of a serial line and, if HUPCL is set, to de-assert it on the last close().
To change the operation of this feature, issue the set on the
/platform/platform/kernel/drv/stc.conf parameter flags field bit
DTR_ASSERT.

Many default parameters of the serial and parallel ports can be changed using
the /platform/platform/kernel/drv/stc.conf file. The format of a line in
the stc.conf file is:

device_tag=token [=value] [: token [=value]]

For serial ports, the device_tag is stc_n, where n is between 0 and the maximum
number of serial ports used by the driver. The token and parameters that
follow it apply to both the /dev/term/n entries and /dev/cua/n entries.

For parallel ports, the device_tag is stc_fn, where n is between 0 and the
number of parallel ports driven by stc.
The *token* [=*value*] specifies a *token*, and if the *token* takes a *value*, the *value* to assigned. Tokens that don’t take a value are considered boolean. If boolean tokens don’t appear in the stc.conf file, they will be cleared by the driver. If these tokens appear in the stc.conf file, they will be set by the driver.

Tokens that take parameters must have a parameter specified in the *token*=*value* couplet in the stc.conf file. If no parameter or an invalid parameter is specified, the driver will ignore the token and revert to using the driver’s default value.

### Tokens for Serial Ports

Valid boolean tokens for serial ports are:

- **soft_carrier** - Default value, enables the soft carrier on the specified line. When the soft carrier is set, transitions on the carrier detect line will be ignored. Use `drt_assert` to clear this value.

- **dtr_assert** - Causes the DTR to be asserted on the next open of the port.

- **dtr_force** - Causes DTR to be continuously asserted. It overrides any other DTR operations and ioctl calls.

- **dtr_close** - Use alternate semantics when dealing with DTR in close. If this is clear, DTR will drop on the close of the port. If this is set, DTR will not drop on `close()` if `TS_SOFTCAR` (see `termios(7I)`) is set in the `t_flags`.

- **cflow_flush** - Flush any data being held off by remote flow control on `close()`.

- **cflow_msg** - Display a message on the console if data transmission is stalled due to remote flow control blocking the transfer in `close()`.

- **instantflow** - If transmission is stopped by software flow control and the flow control is disabled via an `ioctl()` call, the transmitter will be enabled immediately.

Valid tokens requiring values are:

- **drain_size** - The size of STREAMS buffers allocated when passing data from the receive interrupt handler upstream.

- **hiwater, lowwater** - The high water and low water thresholds in the receive interrupt handler 1024 byte buffer.

- **rtpr** - The inter-character receive timer.
The UART receive fifo threshold.

For the value-carrying tokens for serial ports:

<table>
<thead>
<tr>
<th>token</th>
<th>default value</th>
<th>min value</th>
<th>max value</th>
</tr>
</thead>
<tbody>
<tr>
<td>hiwater</td>
<td>1010 bytes</td>
<td>2 bytes</td>
<td>1022 bytes</td>
</tr>
<tr>
<td>lowwater</td>
<td>512 bytes</td>
<td>2 bytes</td>
<td>hiwater-2 bytes</td>
</tr>
<tr>
<td>drain_size</td>
<td>64 bytes</td>
<td>4 bytes</td>
<td>1024 bytes</td>
</tr>
<tr>
<td>rptr</td>
<td>18 millisecs</td>
<td>1 millisecs</td>
<td>255 millisecs</td>
</tr>
<tr>
<td>rxfifo</td>
<td>4 bytes</td>
<td>1 bytes</td>
<td>8 bytes</td>
</tr>
</tbody>
</table>

Valid boolean tokens for parallel ports are

- **paper_out** - If set, the PAPER OUT signal from the port is monitored. If clear, the signal is ignored.
- **error** - Monitor the ERROR signal from the port. Ignore the signal if clear.
- **busy** - Monitor the BUSY signal from the port. Ignore the signal if clear.
- **select** - Monitor the SELECT, or ON LINE, signal from the port. Ignore the signal if clear.
- **pp_message** - If this token is clear, a console message will be printed when any of the above four enabled conditions are detected, and another when the condition is cleared. If set, a console message will be printed every 60 seconds until the condition is cleared.
- **pp_signal** - If this token is set, the parallel port’s controlling process will get a PP_SIGTYPE signal whenever one of the above four conditions is detected. PP_SIGTYPE is defined in stcio.h, which is available to the user.

Valid tokens requiring parameters for the parallel ports are

- **ack_timeout** - The amount of time in seconds to wait for an ACK from the port after asserting STROBE and transferring a byte of data.
- **error_timeout** - Amount of time in seconds to wait for an error to go away.
**PARALLEL PORT PARAMETERS**

The default values of certain parallel port parameters that govern data transfer between the SPC/S board and the device attached to the parallel port will usually work well with most devices; however, some devices don’t strictly adhere to the IBM PS/2-compatible (Centronics-compatible) data transfer and device control/status protocol, and may require modification of one or more of the default parallel port parameters. Some printers, for example, have non-standard timing on their SELECT line, which manifests itself if you start sending data to the printer and then take it off line; when you put it back on line, the printer will not assert it's SELECT line until after the next character is sent to the printer. Since the stc driver will not send data to the device if it’s SELECT line is de-asserted, a deadlock condition occurs. To remedy this situation, you can change the default signal list that the stc driver monitors on the parallel port by removing the SELECT signal from the list. This can be done either through the /platform/platform/kernel/drv/stc.conf configuration file or programmatically through the STC_SPPC ioctl() call.

<table>
<thead>
<tr>
<th>token</th>
<th>default value</th>
<th>min value</th>
<th>max value</th>
</tr>
</thead>
<tbody>
<tr>
<td>strobe_width</td>
<td>2 microsecs</td>
<td>1 microsecs</td>
<td>30 microsecs</td>
</tr>
<tr>
<td>data_setup</td>
<td>2 microsecs</td>
<td>0 microsecs</td>
<td>30 microsecs</td>
</tr>
<tr>
<td>ack_timeout</td>
<td>60 seconds</td>
<td>5 seconds</td>
<td>7200 seconds</td>
</tr>
<tr>
<td>error_timeout</td>
<td>5 seconds</td>
<td>1 seconds</td>
<td>480 seconds</td>
</tr>
<tr>
<td>busy_timeout</td>
<td>10 seconds</td>
<td>0 seconds</td>
<td>7200 seconds</td>
</tr>
</tbody>
</table>

**LOADABLE ISSUES**

If you try to unload the driver, and one or more of the ports on one or more of the SPC/S boards is in use (for example, open()) by a process, the driver will not be unloaded, and all lines on all SPC/S boards, with the exception of the control ports, will be marked with an open inhibit flag to prevent further opens until the driver is successfully unloaded.

**ERRORS**

An open() will fail with errno set to:
ENXIO  The unit being opened does not exist.

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

EPERM  The control port for the board was opened by a process whose uid was not root.
       An ioctl() will fail with errno set to:
ENOSR  A STREAMS data block could not be allocated to return data to the caller.

EINVAL  An invalid value was passed as the data argument to the ioctl() call or an invalid argument or op-field was passed in one of the driver-specific ioctl()'s.

EPERM  An STC_GSTATS ioctl() was requested by a process whose uid was not root.

ENOTTY  An unrecognized ioctl() command was received.

FILES

/dev/term/[00-3f]
/dev/ttyy[00-3f]
       hardwired and dial-in tty lines
/dev/cua/[00-3f]
/dev/ttyz[00-3f]
       dial-out tty lines
/dev/printers/[0-7]
/dev/stclp[0-7]
       parallel port lines
/dev/stc[0-7]
control port

/platform/platform/kernel/drv/stc.conf

driver configuration file

/usr/include/sys/stcio.h

header file with ioctl()’s supported by this driver

SEE ALSO
tip(1), uucp(1C), pmadm(1M), termios(3), ldterm(7M), termio(7l),
termiox(7l), ttcompat(7M), allocb(9F), bufcall(9F), kmem_zalloc
(9F)

DIAGNOSTICS
All diagnostic messages from the driver appear on the system console. There
are three severity levels of messages displayed:

FATAL- the device driver does not get loaded and any SPC/S boards
installed in the system are inaccessible (usually occurs
during the process of modload’ing the driver).

ERROR- some condition has caused the normal operation of the
board and/or device driver to be disrupted; data loss may or
may not occur; this class of message might indicate an
impending hardware failure.

ADVISORY- the device driver has detected a condition that may be of
interest to the user of the system; usually this is a transient
condition that clears itself.

Messages During
Initialization Of
Driver/Board.

stc_attach: can’t allocate memory for unit structs

FATAL. kmem_zalloc() failed to allocate memory for the driver’s internal
data structures.

stc_attach: board revision undeterminable

FATAL. The driver did not get a hardware revision level from the board’s
onboard FCode PROM.

stc_attach: board revision 0x%x not supported by driver.
FATAL. This revision of the board is not supported by the driver.

stc_attach: oscillator revision undeterminable

FATAL. The driver did not get an oscillator revision level from the board’s onboard FCode PROM.

stc_attach: weird oscillator revision (0x%x), assuming 10Mhz

ADVISORY. The board’s onboard FCode PROM returned an unanticipated baud-rate oscillator value, so the driver assumes that a 10Mhz oscillator is installed.

stc_attach: error initializing stc%d

FATAL. An error occurred while trying to initialize the board; perhaps a memory access failed.

stc_attach: bad number of interrupts: %d

FATAL. An incorrect number of interrupts was read from the board’s onboard FCode PROM.

stc_attach: bad number of register sets: %d

FATAL. An incorrect number of register sets was read from the board’s onboard FCode PROM.

stc_init: stc%d GIVR was not 0x0ff, was: 0x%x

FATAL. The cd-180 8-channel UART failed to initialize properly, or a memory fault occurred while trying to access the chip.

cd180_init: stc%d GIVR was not 0x0ff, was: 0x%x

FATAL. The cd-180 8-channel UART failed to initialize properly, or a memory fault occurred while trying to access the chip.

stc%d: board revision: 0x%x should be updated

ADVISORY. Two versions of the FCode PROM on the SPC/S card have been released; V1.0 (0x4) and V1.1 (0x5). The V1.1 PROM fixes some incompatibilities between the V1.0 FCode PROM (on the SPC/S) and the
V2.0 OpenBOOT PROM (on your system) and is required on an SPC/S card to be used in a system running Solaris 2.X.

**stc%d:** system boot PROM revision V%d.%d should be updated

**ADVISORY:** Your system’s BOOT PROM should be updated to at least V1.3 because prior versions of the BOOT PROM did not map the SBus interrupt levels that the SPC/S uses correctly.

**Messages Related To The Serial Port.**

**SET_CCR:** CCR timeout

**ERROR:** the cd-180’s CCR register did not return to zero within the specified timeout period after it was issued a command

**PUTSIL0:** unit %d line %d soft silo overflow

**ERROR:** The driver’s internal receive data silo for the enunciated line has overflowed because the system has not gotten around to pulling data out of the silo; check that you are using the correct flow control; all data in the silo is flushed;

*this message may also frequently appear due to a hardware crosstalk problem that was fixed in later releases*

**stc_rcvex:** unit %d line %d receiver overrun, char: 0x%x

**ERROR:** The driver could not get around to service the cd-180 receive data interrupt before the cd-180’s receive data fifo filled up;

*this message may also frequently appear due to a hardware crosstalk problem that was fixed in later releases*

**stc_drainsilo:** unit %d line %d can’t allocate streams buffer

**ERROR:** The driver could not get a STREAMS message buffer from `bufcall(9F)`; all data in the driver’s receive data silo is flushed.

**stc_drainsilo:** unit %d line %d punting put retries

**ERROR:** After trying several times to send data down the stream from the driver to the application and finding the path blocked, the driver gives up; all data in the driver’s receive data silo is flushed.

**stc_modem:** unit %d line %d interesting modem control

**ADVISORY:** The cd-180 posted a modem control line change interrupt, but upon examination by the driver, no modem control lines had changed state.
since the last time a scan was conducted; if you see this problem frequently, it is likely that your data cables are either too long or picking up induced noise.

Messages Related To The Parallel Port.

ppc_stat: unit %d PAPER OUT

ADVISORY. The device connected to the parallel port on the enumerated board has signalled that it is out of paper (PAPER OUT line asserted).

ppc_stat: unit %d PAPER OUT condition cleared

ADVISORY. The previously-detected paper out condition has been cleared by the device connected to the parallel port on the enumerated board (PAPER OUT line de-asserted).

ppc_stat: unit %d OFFLINE

ADVISORY. The device connected to the parallel port on the enumerated board has signaled that it is off-line (SLCT line de-asserted).

ppc_stat: unit %d OFFLINE condition cleared

ADVISORY. The previously-detected off line condition has been cleared by the device connected to the parallel port on the enumerated board (SLCT line asserted).

ppc_stat: unit %d ERROR

ADVISORY. The device connected to the parallel port on the enumerated board has signalled that it has encountered an error of some sort (ERROR line asserted).

ppc_stat: unit %d ERROR condition cleared

ADVISORY. The previously-detected error condition has been cleared by the device connected to the parallel port on the enumerated board (ERROR line de-asserted).

ppc_acktimeout: unit %d ACK timeout

ERROR. The ACK line from the device connected to the parallel port did not assert itself within the configurable timeout period; check to be sure that the device is connected and powered on.
**Devices**

**ppc_acktimeout:** unit `%d` BUSY timeout

ERROR. The BUSY line from the device connected to the parallel port did not de-assert itself within the configurable timeout period; check to be sure that the device is connected and powered on.

**ppc_int:** unit `%d` stray interrupt

ADVISORY. The parallel port controller (ppc) chip generated an interrupt while the device was closed; this was unexpected and if you see it frequently, your parallel cable might be picking up induced noise causing the ppc to generate an unwanted interrupt, or this could indicate that the ppc might have an internal problem.

**ppc_acktimeout:** unit `%d` can’t get pointer to read q

ERROR. Somehow the driver’s internal ppc data structure became corrupted; this should not happen.

**ppc_acktimeout:** unit `%d` can’t send M_ERROR message

ERROR. The driver can’t send an M_ERROR STREAMS message to the application; this should not happen either.

**ppc_signal:** unit `%d` can’t get pointer to read q

ERROR. Somehow the driver’s internal ppc data structure became corrupted; this should also not happen.

**ppc_signal:** unit `%d` can’t send M_PCSIG(PP_SIGTYPE 0x%x) message

ERROR. The driver can’t send an M_PCSIG STREAMS message to the application (which could cause a signal to be posted); this should also not happen either.

**Messages Related To STREAMS Processing.**

**stc_wput:** unit `%d` trying to M_STARTI on ppc or control device

ADVISORY. An M_STARTI STREAMS message was sent to the parallel port or the board control device; this should only happen if an application explicitly sends this message.

Last modified 17 Mar 1998 SunOS 5.7 454
Messages Related To Serial Port Control.

stc_ioctl: unit %d line %d can't allocate streams buffer for ioctl

ERROR. The driver could not get a STREAMS message buffer from bufcall() for the requested ioctl; the ioctl will not be executed.

stc_ioctl: unit %d line %d can't allocate STC_DCONTROL block

ERROR. The driver could not allocate a data block from allocb() for the STC_DCONTROL return value; the ioctl does not get executed.

stc_ioctl: unit %d line %d can't allocate STC_GPPC block

ERROR. The driver could not allocate a data block from allocb() for the STC_GPPC return value; the ioctl does not get executed.

stc_ioctl: unit %d line %d can't allocate TIOCMGET block

ERROR. The driver could not allocate a data block from allocb() for the TIOCMGET return value; the ioctl does not get executed.

stc_vdcmd: unit %d cd-180 firmware revision: 0x%x

ADVISORY. The firmware revision level of the cd-180, displayed when the driver is first loaded.
NAME  stp4020 – STP 4020 PCMCIA Adapter

DESCRIPTION  The STP 4020 PCMCIA Adapter provides for two PCMCIA PC Card sockets. The stp4020 adapter driver provides an interface between the PCMCIA sockets and the PCMCIA nexus. The driver supports both the Voyager PCMCIA sockets and the Sun PCMCIA Interface/Sbus card. Direct access to the PCMCIA hardware is not supported. The driver exists solely to support the PCMCIA nexus.

FILES  
/kernel/drv/stp4020 stp4020 driver.

SEE ALSO  pcmcia(4)
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(2)) 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 ioctl, 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:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I_PUSH</td>
<td>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:</td>
</tr>
<tr>
<td>EINVAL</td>
<td>Invalid module name.</td>
</tr>
<tr>
<td>EFAULT</td>
<td>arg points outside the allocated address space.</td>
</tr>
<tr>
<td>ENXIO</td>
<td>Open routine of new module failed.</td>
</tr>
<tr>
<td>ENXIO</td>
<td>Hangup received on fildes.</td>
</tr>
</tbody>
</table>
Ioctl Requests

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.

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 STREAMS 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:

```c
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:
**Ioctl Requests**

**EINV AL**  
*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:

- **EINV AL**  
  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.

- **EINV AL**  
  *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:

```c
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.

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_u scalar_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_u scalar_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_u scalar_t, specifies the offset from the start of the control buffer where I_FDINSERT will store a t_u scalar_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_FDIINSERT 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.

At most one I_STR can be active on a stream. Further I_STR calls will block until the active I_STR completes at the STREAM head. The default timeout interval for these requests is 15 seconds. The O_NDELAY and O_NONBLOCK (see open(2)) flags have no effect on this call.

To send requests downstream, arg must point to a strioctl structure which contains the following members:

```c
int ic_cmd;
int ic_timeout;
int ic_len;
char *ic_dp;
```

ic_cmd is the internal ioctl command intended for a downstream module or driver and ic_timeout 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
ioctl Requests

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).

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_timeout 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.

Last modified 9 Dec 1997

SunOS 5.7 466
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(2)`) 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`. 
Ioctl Requests

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:

```c
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 a str_list structure that has the following members:

```c
int sl_nmods;
struct str_mlist *sl_modlist;
```

The str_mlist structure has the following member:

```c
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.
Ioctl Requests

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:

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:

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:

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 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:

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.

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 multiplexer.

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.

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.
EINV AL  \(\text{arg}\) is an invalid multiplexor ID number or \(\text{fildes}\) is not the STREAM on which the \_I\_LINK that returned \(\text{arg}\) was performed.

EINV AL  \(\text{fildes}\) 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 \(\text{fildes}\). In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, \_I\_UNLINK will fail with \text{errno} set to the value in the message.

\_I\_PLINK

Connects two streams, where \(\text{fildes}\) is the file descriptor of the stream connected to the multiplexing driver, and \(\text{arg}\) is the file descriptor of the STREAM connected to another driver. The STREAM designated by \(\text{arg}\) 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 \(\text{fildes}\) 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, \text{errno} is set to one of the following values:

ENXIO  Hangup received on \(\text{fildes}\).

ETIME  Time out before acknowledgement message was received at the STREAM head.

EAGAIN  Unable to allocate STREAMS storage to perform the \_I\_PLINK.

EBADF  \(\text{arg}\) is not a valid, open file descriptor.

EINV AL  \(\text{fildes}\) does not support multiplexing.

EINV AL  \(\text{arg}\) is not a STREAM or is already linked under a multiplexor.
EINV AL  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.

EINV AL  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.

EINV AL  Invalid multiplexor ID number.

EINV AL  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(2), close(2), fcntl(2), getmsg(2), ioctl(2), open(2), poll(2), putmsg(2), read(2), write(2), signal(3C), signal(5), pipemod(7M)

STREAMS Programming Guide
NAME
sxp – Rockwell 2200 SNAP Streams Driver

SYNOPSIS
/dev/sxp

DESCRIPTION
The sxp (also known as the SNAP )driver is a loadable, clonable, STREAMS driver that supports the connectionless Data Link Provider Interface (dlpi(7P)) over one or more FDDI adapters (Rockwell 2200 Series). The cloning character-special devices (/dev/sxp, /dev/snap, /dev/llc, /dev/mac) are used to access the 2200 Series adapter(s). The /dev/sxp device is equivalent to /dev/snap. /dev/sxp is used so that the name SXP will show up in ifconfig. All messages transmitted on a SNAP device have the 802.2 LLC and Sub-Network Access Protocol (SNAP) and the FDDI MAC headers (RFC-1188) prepended. For an LLC device, the LLC and MAC headers are prepended, and for a MAC device only the MAC header is prepended. Received FDDI frames are delivered to the appropriate open device. In response to a DL_INFO_REQ, the SNAP driver returns the following values in the DL_INFO_ACK primitive:
- The maximum SDU is 4500.
- The minimum SDU is 0.
- The DLSAP address length is 8 (always true in the Solaris environment).
- The address offset is 0 (prior to being attached).
- The MAC type is DL_FDDI.
- The sap length value is -2, which indicates that within the DLSAP address, the physical address component is followed immediately by a 2-byte service access point (SAP) component.
- The service mode is DL_CLDLS.
- The quality of service (QOS) fields are 0, because optional QOS is not supported.
- The provider style is DL_STYLE2.
- The version is DL_VERSION_2.
- The broadcast address value is the IEEE broadcast address (FF:FF:FF:FF:FF:FF).

Because the SNAP driver is a "style 2" Data Link Service provider, an explicit DL_ATTACH_REQ message from the user is required to associate the opened stream with a particular network device (that is, ppa). The dl_ppa field within the DL_ATTACH_REQ indicates the instance (unit) number of the network device. If no currently attached ppa has the same instance number and there are no unattached ppas available, the driver returns an error (DL_ERROR_ACK). Once in the DL_ATTACHED state, a DL_BIND_REQ is required to associate a particular SAP with the stream.

Once in the DL_ATTACHED state, a DL_BIND_REQ is required to associate a particular Service Access Point (SAP) with the stream. For the sap field within the DL_BIND_REQ, valid values are in the range [0-0xFFFF]. Values for 0-0xFF will give LLC 802.2 service without SNAP encapsulation, unless a later DL_HIERARCHIAL_BIND DL_SUBS_BIND_REQ is made. Values from 0x100-0xFFFF will give LLC 802.2 with SNAP encapsulation without the need for a DL_SUBS_BIND_REQ. Note that DL_HIERARCHIAL_BIND class
DL_SUBS_BIND_REQs are only supported on streams bound to the 0xAA SAP. After successful completion of the DL_BIND_REQ, the ppa is initialized and the stream is ready for use. In addition to the DL_HIERARCHIAL_BIND class of DL_SUBS_BUD_REQ, the DL_PEER_BIND class can be used to bind multiple SAPs with a stream.

Frames may be transmitted on the FDDI ring by sending DL_UNITDATA_REQ messages to the SNAP driver. The DLSAP address contained within the DL_UNITDATA_REQ must consist of both the SAP and physical (FDDI) components. For a SNAP device, the SAP portion of the DLSAP address is placed in the EtherType field of the 802.2 SNAP header. The DSAP and SSAP fields of the 802.2 LLC header are both set to the value 170, indicating a SNAP message and a MAC frame_type of LLC. For an LLC device, the SAP portion of the DLSAP address is placed in the DSAP field of the 802.2 LLC header. The SSAP field is set to the SAP bound to the stream. The MAC frame_type is LLC. For a MAC device, the SAP portion of the DLSAP address is placed in the frame_control field of the MAC header. Received FDDI frames are routed up the correct stream(s) as DL_UNITDATA_IND messages (containing the DLSAP address). The stream(s) are found by:

1. Comparing the EtherType field of the SNAP header with the bound SAP of all the SNAP streams
2. Comparing the DSAP field of the LLC header with the bound SAP of all the LLC streams
3. Comparing the frame_control field of the MAC header with the bound SAP of all the MAC streams.

If necessary, messages are duplicated. In addition to the mandatory connectionless DLPI message set, the driver also supports the following primitives: DL_ENABMULTI_REQ, DL_DISABMULTI_REQ, DL_PROMISCON_REQ, DL_PROMISCOFF_REQ, DL_PHYS_ADDR_REQ.

The DL_ENABMULTI_REQ and DL_DISABMULTI_REQ primitives enable or disable reception of individual multicast group addresses. Using these primitives, a set of multicast group addresses may be iteratively created and modified on a per-stream basis. These primitives are accepted by the driver in any state following a successful DL_ATTACH_REQ. The DL_PROMISCON_REQ and DL_PROMISCOFF_REQ primitives (with the DL_PROMISC_PHYS flag set in the dl_level field) enable or 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 or disables reception of all sap values. When used with the DL_PROMISC_MULTI flag (set), this enables or disables reception of all multicast group addresses. The affect of each primitive is always on a per-stream basis, and is independent of the other sap and physical level configurations on this stream (or other streams). In the DL_PHYS_ADDR_ACK message, the DL_PHYS_ADDR_REQ primitive returns the...
6-octet FDDI address (in canonical form) currently associated with the stream. This primitive is valid only in states following a successful DL_ATTACH_REQ. The driver also supports the following ioctl(1) controls: DLIOCRAW, SL_RAW, SL_DATA_ENABLE, SL_DATA_DISABLE, and DRV_CONFIG. As defined by Solaris, the DLIOCRAW ioctl puts the stream into raw mode, which causes the driver to send the full MAC-level packet up the stream in an M_DATA message, instead of transforming it to the DL_UNITDATA_IND form. On this stream, the driver will also accept formatted M_DATA messages for transmission. To disable raw mode, the stream must be closed. The DLIOCRAW ioctl requires no arguments. As defined by Rockwell, the SL_RAW ioctl puts the stream into raw mode, similar to the DLIOCRAW ioctl except that the frame-type field of the MAC header is considered to be a long word instead of a byte, preserving alignment. The SL_RAW ioctl requires no arguments. As defined by Rockwell, the SL_DATA_ENABLE and SL_DATA_DISABLE ioctl's enable or disable the transmission of data on the stream. By default, transmission is enabled. The SL_DATA_ENABLE and SL_DATA_DISABLE ioctl's require no arguments.

FILES
/dev/sxp
SXP special character device

/kernel/drv/sys_core
SXP loadable module

/kernel/drv/sxp.conf
SXP configuration file

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO attributes(5), dlpi(7P), rns_smt(7D)
NAME tcp, TCP – Internet Transmission Control Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_STREAM, 0);

t = t_open("/dev/tcp", 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), 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 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)` 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 address and TCP port number bound with the `bind(3N)` system call after the socket is created. By default, TCP sockets are active. A passive socket is created by calling the `listen(3N)` 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(3N)` system call. Active sockets use the `connect(3N)` call after binding to initiate connections.

By using the special value `INADDR_ANY`, 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 address will be bound at connection time to the address of the network interface used to service the connection.

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(3N) and tested with getsockopt(3N). The option level for the setsockopt() call is the protocol number for TCP, available from getprotobyname(3N).

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).

TCP provides an urgent data mechanism, which may be invoked using the out-of-band provisions of send(3N). The caller may mark one byte as “urgent” with the MSG_OOB flag to send(3N). 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, as would be the case if the remote machine crashed, the connection is closed and an error is returned to the user. If the remote machine reboots or otherwise loses state information about a TCP connection, the connection is aborted and an error is returned to the user.

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).

Turn on the window scale option in one of the following ways:

1. 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.

2. 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 0.

3. 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 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 time stamp 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 \texttt{tcp\_tstamp\_if\_wscale} is 0, the time stamp option will not be set when connecting to a remote system. The default for \texttt{tcp\_tstamp\_if\_wscale} is 0.

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 \textbf{HIGHLY} recommended that you set sequence number generation parameters to be 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 \texttt{/etc/init.d/inetinit} script contains commands which configure initial sequence number generation. The script reads the value contained in the configuration file \texttt{/etc/default/inetinit} to determine which method to use.

The \texttt{/etc/default/inetinit} file is an unstable interface, and may change in future releases.

\textbf{SEE ALSO}

\texttt{ndd(1M), ioctl(2), read(2), write(2), accept(3N), bind(3N),
connect(3N), getprotobyname(3N), getsockopt(3N), listen(3N),
send(3N), inet(7P), ip(7P)}


\textbf{DIAGNOSTICS}

A socket operation may fail if:

\textbf{EISCONN} A \texttt{connect()} operation was attempted on a socket on which a \texttt{connect()} operation had already been performed.

\textbf{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.
NAME tcx – 24-bit SBus color memory frame buffer

SYNOPSIS SUNW,tcx@sbus-slot,offset:tcx

DESCRIPTION tcx is a 8/24-bit color frame buffer and graphics accelerator, with 8-bit colormap and overlay/enable planes. It provides the standard frame buffer interface defined in fbio(7I). sbus-slot is the SBus slot number. (See sbus(4) for more information.) offset is the device offset. X is the kernel-assigned device number.

APPLICATION PROGRAMMING INTERFACE tcx has two control planes which define how the underlying pixel is displayed. The display modes are 8-bit (8 bits taken from low-order 8 bits of pixel) through a colormap; 24-bit through a gamma-correction table; 24-bit through the colormap; or 24-bit direct. The colormap is shared by both 24-bit and 8-bit modes.

The tcx has registers and memory that may be mapped with mmap(2).

There is an 8-bit only version of tcx which operates the same as the 24-bit version, except that the 24-bit-related mappings can not be made.

IOCTLS tcx accepts the following ioctl(2) calls, defined in <sys/fbio.h> and <sys/visual_io.h>, and implemented as described in fbio(7I).

<table>
<thead>
<tr>
<th>FBIOGATTR</th>
<th>FBIOGCURSOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>FBIOGTYPE</td>
<td>FBIOSCURPOS</td>
</tr>
<tr>
<td>FBIOPUTCMAP</td>
<td>FBIOGCURPOS</td>
</tr>
<tr>
<td>FBIOGETCMAP</td>
<td>FBIOGCMAX</td>
</tr>
<tr>
<td>FBIOSATTR</td>
<td>FBIOGXINFO</td>
</tr>
<tr>
<td>FBIOSVIDEO</td>
<td>FBIOMONINFO</td>
</tr>
<tr>
<td>FBOGVIDEO</td>
<td>FBOVROFFSET</td>
</tr>
<tr>
<td>FBOIOVERTICAL</td>
<td>VIS_GETIDENTIFIER</td>
</tr>
<tr>
<td>FBIOCURSOR</td>
<td></td>
</tr>
</tbody>
</table>

VIS_GETIDENTIFIER returns "SUNW,tcx".

Emulation mode (FBIOGATTR, FBiosATTR) may be either FBTYPER_SUN3COLOR or FBTYPER_MEMCOLOR. Set emulation mode to 21 (FBTYPEMALEFTPLUSEONE) to turn emulation off. Changes to emulation mode (via FBIOSSATTR) take place immediately. Emulation may be turned off manually by setting emu_type field of the fbsattr structure to 21. Emulation mode is reset to default on reboot.

Last modified 1 Jan 1997  SunOS 5.7  484
FBIOPUTCMAP returns immediately, although the actual colormap update may be delayed until the next vertical retrace. If vertical retrace is currently in progress, the new colormap takes effect immediately.

FBIOGETCMAP returns immediately with the currently-loaded colormap, unless a colormap write is pending (see above), in which case it waits until the colormap is updated before returning. This may be used to synchronize software with colormap updates.

The size and linebytes values returned by FBIOGATTR, FBIOGTYPE and FBIOGXINFO are the sizes of the 8-bit framebuffer. The proper way to compute the size of a framebuffer mapping is:

\[
\text{size} = \text{linebytes} \times \text{height} \times \text{bytes per pixel}
\]

The information returned in the dev_specific field by the FBIOGATTR ioctl is as follows:

dev_specific[0] is the tcx capabilities mask:

<table>
<thead>
<tr>
<th>Name</th>
<th>Hex Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>STIP_ALIGN</td>
<td>0xf</td>
<td>stipple alignment constraint</td>
</tr>
<tr>
<td>C_PLANES</td>
<td>0xf0</td>
<td># of control planes</td>
</tr>
<tr>
<td>BLIT_WIDTH</td>
<td>0xf00</td>
<td>maximum blit width</td>
</tr>
<tr>
<td>BLIT_HEIGHT</td>
<td>0xf000</td>
<td>maximum blit height</td>
</tr>
<tr>
<td>STIP_ROP</td>
<td>0x10000</td>
<td>stipple-with-rop supported</td>
</tr>
<tr>
<td>BLIT_ROP</td>
<td>0x20000</td>
<td>blit-with-rop supported</td>
</tr>
<tr>
<td>24_BIT</td>
<td>0x40000</td>
<td>24-bit support</td>
</tr>
<tr>
<td>HW_CURSOR</td>
<td>0x80000</td>
<td>hardware cursor</td>
</tr>
<tr>
<td>PLANE_MASK</td>
<td>0x100000</td>
<td>plane mask support for 8-bit stipple</td>
</tr>
</tbody>
</table>

dev_specific[1] is the kernel address for 8-bit mapping. This is useful only to other device drivers, and should not be used outside the kernel.
**FILES**

/dev/fbs/tcx  device special file
/dev/fb      default frame buffer

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>SPARCstation 4, SPARCstation 5</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`ioctl(2), mmap(2), sbus(4), attributes(5), fbio(7I)`
NAME
termio – general terminal interface

SYNOPSIS
#include <termio.h>

ioctl(int fildes, int request, struct termio *arg);

#include <termios.h>

ioctl(int fildes, int request, int arg);

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(3) 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, users’ 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 getsid(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

487 SunOS 5.7 Last modified 6 Mar 1998
ioctl Requests

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.

The operating system will not normally send SIGTSTP, SIGTTIN, or SIGTTOU signals to a process that is a member of an orphaned process group.

These are process groups which do not have a member with a parent in another process group that is in the same session and therefore shares the same controlling terminal. When a member's orphaned process group attempts to access its controlling terminal, errors will be returned. Since there is no process to continue it if it should stop.

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 will be 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 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 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 be suspended 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; consequently, 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

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 * 0.1 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:

Last modified 6 Mar 1998

SunOS 5.7 490
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(5)`).

**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.
ioctl Requests

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ERASE</strong></td>
<td>(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.</td>
</tr>
<tr>
<td><strong>WERASE</strong></td>
<td>(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.</td>
</tr>
<tr>
<td><strong>KILL</strong></td>
<td>(Control-u or ASCII NAK) deletes the entire line, as delimited by a NL, EOF, EOL, or EOL2 character.</td>
</tr>
<tr>
<td><strong>REPRINT</strong></td>
<td>(Control-r or ASCII DC2) reprints all characters, preceded by a newline, that have not been read.</td>
</tr>
<tr>
<td><strong>EOF</strong></td>
<td>(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.</td>
</tr>
<tr>
<td><strong>NL</strong></td>
<td>(ASCII LF) is the normal line delimiter. It cannot be changed or escaped.</td>
</tr>
<tr>
<td><strong>EOL</strong></td>
<td>(ASCII NULL) is an additional line delimiter, like NL. It is not normally used.</td>
</tr>
<tr>
<td><strong>EOL2</strong></td>
<td>is another additional line delimiter.</td>
</tr>
<tr>
<td><strong>SWITCH</strong></td>
<td>(Control-z or ASCII EM) is used only when sh1 layers is invoked.</td>
</tr>
<tr>
<td><strong>SUSP</strong></td>
<td>(Control-z or ASCII SUB) generates a SIGTSTP signal. SIGTSTP stops all processes in the foreground process group for that terminal.</td>
</tr>
<tr>
<td><strong>DSUSP</strong></td>
<td>(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.</td>
</tr>
<tr>
<td><strong>STOP</strong></td>
<td>(Control-s or ASCII DC3) can be used to suspend output temporarily. It is useful with CRT terminals to prevent...</td>
</tr>
</tbody>
</table>
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, SWITCH, 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 `\' 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:

<table>
<thead>
<tr>
<th>Subscript</th>
<th>Symbol</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>VINTR</td>
<td>ETX</td>
</tr>
<tr>
<td>1</td>
<td>VQUIT</td>
<td>FS</td>
</tr>
<tr>
<td>2</td>
<td>VERASE</td>
<td>DEL</td>
</tr>
<tr>
<td>3</td>
<td>VKILL</td>
<td>NAK</td>
</tr>
<tr>
<td>4</td>
<td>VEOF</td>
<td>EOT</td>
</tr>
<tr>
<td>5</td>
<td>VEOL</td>
<td>NUL</td>
</tr>
<tr>
<td>6</td>
<td>VEOLO2</td>
<td>NUL</td>
</tr>
<tr>
<td>7</td>
<td>VSWITCH</td>
<td>NUL</td>
</tr>
<tr>
<td>8</td>
<td>VSTART</td>
<td>DC1</td>
</tr>
<tr>
<td>9</td>
<td>VSTOP</td>
<td>DC3</td>
</tr>
<tr>
<td>10</td>
<td>VSUSP</td>
<td>SUB</td>
</tr>
<tr>
<td>11</td>
<td>VDSUSP</td>
<td>EM</td>
</tr>
<tr>
<td>12</td>
<td>VREPRINT</td>
<td>DC2</td>
</tr>
<tr>
<td>13</td>
<td>VDISCARD</td>
<td>SI</td>
</tr>
<tr>
<td>14</td>
<td>VWERASE</td>
<td>ETB</td>
</tr>
<tr>
<td>15</td>
<td>VLNEXT</td>
<td>SYN</td>
</tr>
<tr>
<td>16-19</td>
<td>reserved</td>
<td></td>
</tr>
</tbody>
</table>

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, \0.

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, X, where X is the data of the byte received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of \377 is given to the application as \377, \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 (notread). 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.
- **ONLNR** 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**

---

Last modified 6 Mar 1998
CR1  
CR2  
CR3  
TABDLY  Select horizontal tab delays:  
TAB0  or tab expansion:  
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 OPPOST 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 OMLCR 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 is set to 0 and no delays are used for CR.
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:

<table>
<thead>
<tr>
<th>Baud rate:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>B0</td>
<td>Hang up</td>
</tr>
<tr>
<td>B50</td>
<td>50 baud</td>
</tr>
<tr>
<td>B75</td>
<td>75 baud</td>
</tr>
<tr>
<td>B110</td>
<td>110 baud</td>
</tr>
<tr>
<td>B134</td>
<td>134 baud</td>
</tr>
<tr>
<td>B150</td>
<td>150 baud</td>
</tr>
<tr>
<td>B200</td>
<td>200 baud</td>
</tr>
<tr>
<td>B300</td>
<td>300 baud</td>
</tr>
<tr>
<td>B600</td>
<td>600 baud</td>
</tr>
<tr>
<td>B1200</td>
<td>1200 baud</td>
</tr>
</tbody>
</table>

Last modified 6 Mar 1998 SunOS 5.7 498
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1800</td>
<td>1800 baud</td>
</tr>
<tr>
<td>B2400</td>
<td>2400 baud</td>
</tr>
<tr>
<td>B4800</td>
<td>4800 baud</td>
</tr>
<tr>
<td>B9600</td>
<td>9600 baud</td>
</tr>
<tr>
<td>B19200</td>
<td>19200 baud</td>
</tr>
<tr>
<td>EXT A</td>
<td>External A</td>
</tr>
<tr>
<td>B38400</td>
<td>38400 baud</td>
</tr>
<tr>
<td>EXT B</td>
<td>External B</td>
</tr>
<tr>
<td>B57600</td>
<td>57600 baud</td>
</tr>
<tr>
<td>B76800</td>
<td>76800 baud</td>
</tr>
<tr>
<td>B115200</td>
<td>115200 baud</td>
</tr>
<tr>
<td>B153600</td>
<td>153600 baud</td>
</tr>
<tr>
<td>B230400</td>
<td>230400 baud</td>
</tr>
<tr>
<td>B307200</td>
<td>307200 baud</td>
</tr>
<tr>
<td>B460800</td>
<td>460800 baud</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSIZE</td>
<td>Character size:</td>
</tr>
<tr>
<td>CS5</td>
<td>5 bits</td>
</tr>
<tr>
<td>CS6</td>
<td>6 bits</td>
</tr>
<tr>
<td>CS7</td>
<td>7 bits</td>
</tr>
<tr>
<td>CS8</td>
<td>8 bits</td>
</tr>
<tr>
<td>CSTPB</td>
<td>Send two stop bits, else one</td>
</tr>
<tr>
<td>CREAD</td>
<td>Enable receiver</td>
</tr>
<tr>
<td>PARENB</td>
<td>Parity enable</td>
</tr>
<tr>
<td>PAR ODD</td>
<td>Odd parity, else even</td>
</tr>
<tr>
<td>HUPCL</td>
<td>Hang up on last close</td>
</tr>
<tr>
<td>CLOCAL</td>
<td>Local line, else dial-up</td>
</tr>
</tbody>
</table>
Ioctl Requests

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.

```c
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.

```c
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.

```c
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.

```c
speed_t ispeed;
if (ispeed == 0) {
    ispeed = termios_p->c_cflag & CBAUD;
    if (termios_p->c_cflag & CBAUDEXT)
        ispeed += (CBAUD + 1);
} else
    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 `CRTSCTS` 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 `CRSXOFF` bits and their interactions are described below.

**Case A:** `CRTSCTS` off, `CRSXOFF` off. In this case the hardware flow control is disabled.

**Case B:** `CRTSCTS` on, `CRSXOFF` 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, `CRSXOFF` 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, `CRSXOFF` 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 `^?`. 

Last modified 6 Mar 1998
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. 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 if ICANON is set, an upper case letter is accepted on input by preceding it with a `\' character, and is output preceded by a `\' character. In this mode, the following escape sequences are generated on output and accepted on input:

For: Use:
` \`
| `!`
|= \^
{ \{
} \)
 ` ``
For example, A is input 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 ECHOPRT).

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:

```c
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 `ioctl`s; it is defined by `sys/termios.h` and includes the following members:

```c
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:

```
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, the modem control lines supported by the hardware can be read, and the modem status lines supported by the hardware can be changed. 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`. Note: This property is assigned during system initialization, therefore 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 **attty(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 requests supported by devices and STREAMS modules providing the termios(3) 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.

**TCSETS W**

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
Ioctl Requests

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.

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. See termios(3) for a description of TCSETPGRP.

TIOCGSID
The argument is a pointer to a pid_t. The session ID of the terminal is fetched and stored in the pid_t.

TIOCSPRSP
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.

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.
Ioctl Requests

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.

FILES  files in or under /dev

SEE ALSO  stty(1), fork(2), getsid(2), ioctl(2), setsid(2), sigaction(2), signal(3C), termios(3), signal(5), 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 uni-directional 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 uni-directional and bi-directional hardware flow control; when bi-directional 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 to each other. 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...
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:

```c
#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 0000008** Enable CD hardware flow control on output.
- **ISxOFF 0000016** 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, bi-directional RTS/CTS flow...
control is selected by setting both the \texttt{RTSXOFF} and \texttt{CTSXON} bits and bi-directional DTR/CTS flow control is selected by setting both the \texttt{DTRXOFF} and \texttt{CTSXON}. Modem control or uni-directional CTS hardware flow control is selected by setting only the \texttt{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 \texttt{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 \texttt{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 \texttt{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 \texttt{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 \texttt{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 \texttt{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 \texttt{x_cflag} fields. If the port is not programmed for external clock generation, \texttt{ISXOFF} is ignored. Output isochronous flow control is supported by appropriate clock source programming using the \texttt{x_cflag} field and enabled at the remote connected device.

The \texttt{x_cflag} field specifies the system treatment of clock modes.

<table>
<thead>
<tr>
<th>\texttt{x_cflag}</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{XMTCLK}</td>
<td>00000007</td>
<td>Transmit clock source:</td>
</tr>
<tr>
<td>\texttt{XCIBRG}</td>
<td>00000000</td>
<td>Get transmit clock from internal baud rate</td>
</tr>
<tr>
<td>\texttt{XCTSET}</td>
<td>00000001</td>
<td>Get transmit clock from transmitter</td>
</tr>
</tbody>
</table>
Ioctl Requests

<table>
<thead>
<tr>
<th>signal</th>
<th>element timing (DCE source) lead, CCITT V.24 circuit 114, EIA-232-D pin 15.</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCRSET</td>
<td>0000002 Get transmit clock from receiver signal element timing (DCE source) lead, CCITT V.24 circuit 115, EIA-232-D pin 17.</td>
</tr>
<tr>
<td>RCVCLK</td>
<td>0000070 Receive clock source:</td>
</tr>
<tr>
<td>RCIIBRG</td>
<td>0000000 Get receive clock from internal baud rate generator.</td>
</tr>
<tr>
<td>RCTSET</td>
<td>0000010 Get receive clock from transmitter signal element timing (DCE source) lead, CCITT V.24 circuit 114, EIA-232-D pin 15.</td>
</tr>
<tr>
<td>RCRSET</td>
<td>0000020 Get receive clock from receiver signal element timing (DCE source) lead, CCITT V.24 circuit 115, EIA-232-D pin 17.</td>
</tr>
<tr>
<td>TSETCLK</td>
<td>0000700 Transmitter signal element timing (DTE source) lead, CCITT V.24 circuit 113, EIA-232-D pin 24, clock source:</td>
</tr>
<tr>
<td>TSETOFF</td>
<td>0000000 TSET clock not provided.</td>
</tr>
<tr>
<td>TSETCBRG</td>
<td>0000100 Output receive baud rate generator on circuit 113.</td>
</tr>
<tr>
<td>TSETCBRG</td>
<td>0000200 Output transmit baud rate generator on circuit 113.</td>
</tr>
<tr>
<td>TSETCTSET</td>
<td>0000300 Output transmitter signal element timing (DCE source) on circuit 113.</td>
</tr>
<tr>
<td>TSETERSET</td>
<td>0000400 Output receiver signal element timing (DCE source) on circuit 113.</td>
</tr>
<tr>
<td>RSETCLK</td>
<td>0000700 Receiver signal element timing (DTE source) lead, CCITT V.24 circuit 128, no EIA-232-D pin, clock source:</td>
</tr>
<tr>
<td>RSETOFF</td>
<td>0000000 RSET clock not provided.</td>
</tr>
<tr>
<td>RSETCRB</td>
<td>0000100 Output receive baud rate generator on circuit 128.</td>
</tr>
<tr>
<td>RSETCTBRG</td>
<td>0000200 Output transmit baud rate generator on circuit 128.</td>
</tr>
<tr>
<td>RSETCTSET</td>
<td>0000300 Output transmitter signal element timing (DCE source) on circuit 128.</td>
</tr>
<tr>
<td>RSETCRSET</td>
<td>0000400 Output receiver signal element timing (DCE source) on circuit 128.</td>
</tr>
</tbody>
</table>

If the XMTCLK field has a value of XCIIBRG 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 RCIIBRG the receive clock is taken from the hardware Internal Baud Rate Generator, as in normal asynchronous transmission.
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 Receiver 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.

The ioctl(2) system calls have the form:

```c
ioctl
    (fd, command, arg)
```

```c
struct termio *
    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(3N)`):

- 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(3N)`):

- TCO_NOPEER: no listener on destination address
- TCO_PEERNOROOMONQ: peer has no room on connect queue
- TCO_PEERBADSTATE: peer in wrong state
TCO_PEERINITIATED
\011peer-initiated disconnect

TCO_PROVIDERINITIATED
\011provider-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(3N)):

TCOO_NOPEER
\011no listener on destination address

TCOO_PEERNOROOMONQ
\011peer has no room on connect queue

TCOO_PEERBADSTATE
\011peer in wrong state

TCOO_PROVIDERINITIATED
\011provider-initiated disconnect

TCO_PEERINITIATED
\011peer-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

Last modified 3 Jul 1990
ticlts(7D) Devices

/dev/ticotsord

SEE ALSO t_rcvdis(3N), t_rcvuderr(3N)
NAME       timod – Transport Interface cooperating STREAMS module

SYNOPSIS   
```c
#include <sys/stropts.h>
```

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:

```c
#include <sys/stropts.h>
-
-
struct strioctl my_strioctl;
-
-
strioctl.ic_cmd = cmd;
strioctl.ic_timeout = 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(2)`, `ioctl(2)`, `errno(3C)`, `tirdwr(7M)`

**STREAMS Programming Guide Transport Interfaces 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(2)`).
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 Transport Interface 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
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:

Messages that represent expedited data generate an error.

All further system calls associated with the stream fail with errno set to EPROTO.

Any data messages with control portions have the control portions associated with the stream fails with errno set to EPROTO.
STREAMS Modules

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
Any zero length data messages are freed by the module and they are not passed onto the module's upstream neighbor.

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(2), getmsg(2), putmsg(2), read(2), write(2), intro(3), streamio(7I), timod(7M)

STREAMS Programming Guide Transport Interfaces Programming Guide
tmpfs(7FS)

NAME
tmpfs – memory based filesystem

SYNOPSIS
#include <sys/mount.h>

mount(special, directory, MS_DATA, "tmpfs", NULL, 0);

DESCRIPTION
tmpfs is a memory based filesystem which uses kernel resources relating to the VM system and page cache as a filesystem. Once mounted, a tmpfs filesystem 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 filesystem that is unmounted will be lost.

tmpfs filesystems can be mounted with the command:

mount -F tmpfs swap directory

Alternatively, to mount a tmpfs filesystem on /tmp at multi-user startup time (and 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 filesystem. Performance improvements are most noticeable when a large number of short lived files are written and accessed on a tmpfs filesystem. 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 filesystem. 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 filesystem 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 filesystem 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 filesystem.
SEE ALSO

df(1M), mount(1M), mount_tmpfs(1M), swap(1M), mmap(2), mount(2), umount(2), vfstab(4)

System Administration Guide, Volume I

DIAGNOSTICS

If tmpfs runs out of space, one of the following messages will be printed to the console.

directory: File system full, swap space limit exceeded
This message is printed 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 filesystem 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 filesystem size is not static and the space available to tmpfs is dependent on the swap space demands of the entire system.
tpf – Platform Specific Module (PSM) for Tricord Systems Enterprise Server Models ES3000, ES4000 and ES5000.

description

The psm is automatically invoked on an ESxxxx platform at system boot time.

files

ATTRIBUTES

see attributes(5) for descriptions of the following attributes:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO attributes(5)
NAME

tr – IBM 16/4 Token Ring Network Adapter device driver

SYNOPSIS

#include <sys/stropts.h>
#include <sys/ethernet.h>
#include <sys/dlpi.h>

DESCRIPTION

The tr token ring driver is a multi-threaded, loadable, clonable, STREAMS
hardware driver supporting the connectionless Data Link Provider Interface,
dlpi(7P), over IBM 16/4 Token Ring adapters and IBM Auto 16/4 Token Ring
adapters. The driver supports installation of both a primary and secondary 16/
4 Adapter within the system. The tr driver provides basic support for the IBM
16/4 adapter and IBM Auto 16/4 Token Ring adapter hardware. Functions
include chip initialization, frame transmit and receive, functional addresses,
and “promiscuous” support, error recovery and reporting, and ring speed
auto-sensing (for the IBM Auto 16/4 Token Ring adapter only).

The cloning, character-special device /dev/tr is used to access all 16/4
adapter devices installed within the system.

The tr driver is a “style 2” Data Link Service provider. All M_PROTO and
M_PCPROTO type messages 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 integer and indicates the corresponding device instance (unit)
number. The unit numbers are assigned sequentially to each board found. The
search order is determined by the order defined in the
/platform/i86pc/kernel/drv/tr.conf file. 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) on 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 4084.
- The minimum SDU is 0.
- The dlsap address length is 7 or 8 bytes.
- The MAC type is DL_TPR.
- The sap length value is −1 or −2, meaning the physical address component
  is followed immediately by a 1 or 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 the IEEE broadcast address (FF:FF:FF:FF:FF:FF). The token ring broadcast address (C0:00:FF:FF:FF:FF) is also supported.

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 tr 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, of which only even values are legal.

In addition to 802.2 service, a “SNAP mode” is also provided by the driver. In this mode, sap values in the range [0x5de-0xffffffff] are used to indicate a request to use “SNAP” mode.

The tr driver DLSAP address format consists of the 6-byte physical token ring address component followed immediately by the 1 or 2-byte sap component, producing a 7 or 8-byte DLSAP address. Applications should not hardcode to this particular implementation-specific DLSAP address format, but should instead use information returned by 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 token ring by sending DL_UNITDATA_REQ messages to the tr driver. The tr driver will route received token ring frames up all open and bound streams that have a sap which matches the sap in the DL_UNITDATA_IND messages. Received token ring 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 and physical (token ring) components.

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 is currently unsupported for this driver.

When used with the DL_PROMISC_SAP flag set, this enables/disables reception of all sap values. When used with the DL_PROMISC_MULTI flag set, this enables/disables reception of all functional 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 token ring 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 token ring address currently associated (attached) to this stream. The credentials of the process which originally opened this stream must be superuser or an EPERM error 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. Once changed, all streams 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 system is rebooted, whichever comes first.

**CONFIGURATION**

The /platform/i86pc/kernel/drv/tr.conf file supports the following options:

- **intr**
  Specifies the IRQ level for the board. Note that if the dip switches for the board are set to use the cascade interrupt, IRQ 2, the IRQ level specified in the configuration file should be IRQ 9.

- **ioaddr**
  Specifies the beginning I/O port address occupied by the board.

- **mtu_size**
  WARNING: Contact your system administrator before modifying the mtu_size.

  This property allows the user to directly set the mtu_size that the adapter will use. If the property is not set, the mtu_size will default to the maximum allowed by the adapter.

  The mtu_size must be a multiple of 8. The minimum allowed value is 96. Values less than the minimum will be
set to 96. Values greater than the maximum will be set to the adapter’s maximum.

Incorrectly setting this value could mean that the token ring card will no longer work on the network.

reg

The first register property specifies the location and size of the board’s BIOS/MMIO area. The second register property specifies the location and size of the board’s shared RAM.

It is important to ensure that there are no conflicts for the board’s I/O port, shared RAM, or IRQ level.

FILES
/dev/tr tr character special device
/platform/i86pc/kernel/drv/tr.conf tr configuration file

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO attributes(5), dlpi(7P)

NOTE
IBM 16/4 Token Ring Network adapters, IBM Auto 16/4 Token Ring adapters, and compatibles are not capable of fully supporting the snoop(1M) program. This limitation is due to the hardware itself and not to a bug in the tr driver or the snoop program.
NAME
trantor – low-level module for Trantor T348 Parallel SCSI host bus adapter

SYNOPSIS
trantor@ioaddr, 0

DESCRIPTION
The trantor module provides low-level interface routines between the common disk/tape I/O subsystem and the Trantor T348 Mini-SCSI-Plus parallel host bus adapter. The Trantor T348 is a SCSI (Small Computer System Interface) host bus adapter that plugs into a parallel port (rather than a bus slot). It offers lower performance than other host bus adapters, but has the advantage of being compatible with notebook computers which do not accept a standard host bus adapter card.

CONFIGURATION
Autoconfiguration code determines whether the adapter is present at the configured address. Autoconfiguration code also determines what types of devices are attached to the adapter.

The trantor.conf configuration file contains no user-configurable items.

FILES
/kernel/drv/trantor.conf trantor device driver configuration file

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO
driver.conf(4), sysbus(4), attributes(5)

NOTES
The trantor driver can support only one T348 adapter per system.

The trantor driver does not support the T348 pass-through parallel port.

The trantor driver does not support concurrent use of SCSI devices and any other device (such as a printer) connected to the same parallel port. All SCSI devices must be closed before any other peripheral devices on the parallel port can be accessed.
NAME
	ttcompat – V7, 4BSD and XENIX STREAMS compatibility module

SYNOPSIS

#include <sys/stream.h>
#include <sys/stropts.h>
#include <sys/ttold.h>
#include <sys/ttcompat.h>
#include <sys/filio.h>

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 TCFLUSH 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>:

```c
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, and reflect the values in the `c_cflag` field of the `termios` structure at a specific time in the past, but are not necessarily reflective of a one-to-one correspondence in functionality. 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 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:

<table>
<thead>
<tr>
<th>sg_flags</th>
<th>c_oflag</th>
</tr>
</thead>
<tbody>
<tr>
<td>BS1</td>
<td>BS1</td>
</tr>
<tr>
<td>FF1</td>
<td>VT1</td>
</tr>
<tr>
<td>CR1</td>
<td>CR2</td>
</tr>
<tr>
<td>CR2</td>
<td>CR3</td>
</tr>
<tr>
<td>CR3</td>
<td>not supported</td>
</tr>
<tr>
<td>TAB1</td>
<td>TAB1</td>
</tr>
<tr>
<td>TAB2</td>
<td>TAB2</td>
</tr>
<tr>
<td>XTABS</td>
<td>TAB3</td>
</tr>
<tr>
<td>NL1</td>
<td>ONLRET|CR1</td>
</tr>
<tr>
<td>NL2</td>
<td>NL1</td>
</tr>
</tbody>
</table>

If previous TIOCLSET or TIOCLBIS ioctl calls have not selected LITOUT or PASS8 mode, and if RAW mode is not selected, then 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.

Parity is not to be generated on output or checked on input. The character size is set to CS8 and the flag is cleared in the `c_cflag` field of the `termios` structure.
Even parity characters are to be generated on output and accepted on input. The flag is set in the c_iflag field of the termios structure, the character size is set to CS7 and the flag is set in the c_cflag field of the termios structure.

Odd parity characters are to be generated on output and accepted on input. The flag is set in the c_iflag field, the character size is set to CS7 and the flags are set in the c_cflag field of the termios structure.

Even parity characters are to be generated on output and characters of either parity are to be accepted on input. The flag is cleared in the c_iflag field, the character size is set to CS7 and the flag is set in the c_cflag field of the termios structure.

The RAW flag disables all output processing (the OPOST flag in the c_oflag field, and the XCASE flag in the c_lflag 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). 8 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_lflag field of the termios structure.

The CRMOD flag turns input RETURN characters into NEWLINE characters, and output and echoed NEWLINE characters to be output as a 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:

\begin{verbatim}
  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
\end{verbatim}

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 \texttt{<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, 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
LCCRTERA ECHOE in the c_lflag field
LTILDE 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
LCTLECH CTLECH in the c_lflag field
Another structure associated with each terminal is the `ltchars` structure which defines control characters for the new 4BSD terminal interface. Its structure is:

```c
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`  `VSUSP`
  - `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. 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.

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.

TIOCFUSH
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.

TIOCBRK
The argument is ignored. The break bit is set for the device.

TIOCCBRK
The argument is ignored. The break bit is cleared for the device.

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.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIOCGETC</td>
<td>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.</td>
</tr>
<tr>
<td>TIOCSETC</td>
<td>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.</td>
</tr>
<tr>
<td>TIOCLGET</td>
<td>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.</td>
</tr>
<tr>
<td>TIOCLBIS</td>
<td>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.</td>
</tr>
<tr>
<td>TIOCLBIC</td>
<td>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.</td>
</tr>
<tr>
<td>TIOCLSET</td>
<td>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.</td>
</tr>
<tr>
<td>TIOCGLTC</td>
<td>The argument is a pointer to an ltchars structure. The values of the appropriate characters in the terminal state are stored in that structure.</td>
</tr>
<tr>
<td>TIOCSLTC</td>
<td>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.</td>
</tr>
<tr>
<td>FIORDCHK</td>
<td>Returns the number of immediately readable characters. The argument is ignored.</td>
</tr>
<tr>
<td>FIONREAD</td>
<td>Returns the number of immediately readable characters in the int pointed to by the argument.</td>
</tr>
</tbody>
</table>
LDSMAP

Calls the function `emsetmap(tp, mp)` if the function is configured in the kernel.

LDGMAP

Calls the function `emgetmap(tp, mp)` if the function is configured in the kernel.

LDNMAP

Calls the function `emunmap(tp, mp)` if the function is configured in the kernel.

The following `ioctl` s 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).

```
TIOCSETD LDOPEN
TIOCGETD LDCLOSE
DIOCSETP LDCHG
DIOCSETP LDSETT
DIIOGETP LDGETT
```

**SEE ALSO**  
`ioctl(2), termios(3), ldterm(7M), termio(7I)`

**NOTES**  
TIOCBRK and TIOCCBRK should be handled by the driver. FIONREAD and FIORDCHK are handled in the stream head.
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
udp, UDP – Internet User Datagram Protocol

SYNOPSIS
#include <sys/socket.h>
#include <netinet/in.h>
s = socket(AF_INET, SOCK_DGRAM, 0);
t = t_open("/dev/udp", O_RDWR);

DESCRIPTION
UDP is a simple datagram protocol which is layered directly above the Internet Protocol (IP). 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(3N) and recv(3N)). If the connect(3N) call is used to fix the destination for future packets, then the recv(3N) or read(2) and send(3N) 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 IP 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(3N) call can be used to set the local address and port number of a UDP socket. The local IP address may be left unspecified in the bind() call by using the special value INADDR_ANY. If the bind() call is not done, a local IP 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.

Options at the IP level may be used with UDP; see ip(7P).

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) error messages received in response to UDP packets it has sent. See icmp(7P). ICMP “source quench” messages are ignored. ICMP “destination unreachable,” “time exceeded” and “parameter problem” messages disconnect the socket from its
Protocols

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.

SEE ALSO
read(2), write(2), bind(3N), connect(3N), recv(3N), send(3N), icmp(7P), inet(7P), ip(7P), tcp(7P)


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.

ENOBUSFS The system ran out of memory for internal data structures.

Last modified 3 Jul 1990

SunOS 5.7 546
NAME
uscsi – user SCSI command interface

SYNOPSIS
#include <sys/scsi/impl/uscsi.h>

ioctl(int fd, 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 */
u_int uscsi_buflen; /* size of i/o to take place */
u_int uscsi_resid; /* resid from i/o operation */
u_char uscsi_cdblen; /* # of valid CDB bytes */
u_char uscsi_rqlen; /* size of uscsi_rqbuf */
u_char uscsi_rqstatus; /* status of request sense cmd */
u_char 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 Flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USCSI_WRITE</td>
<td>Send data to device</td>
</tr>
<tr>
<td>USCSI_SILENT</td>
<td>No error messages</td>
</tr>
<tr>
<td>USCSI_DIAGNOSE</td>
<td>Fail if any error occurs</td>
</tr>
<tr>
<td>USCSI_ISOLATE</td>
<td>Isolate from normal commands</td>
</tr>
<tr>
<td>USCSI_READ</td>
<td>Get data from device</td>
</tr>
<tr>
<td>USCSI_ASYNC</td>
<td>Set bus to asynchronous mode</td>
</tr>
<tr>
<td>USCSI_SYNC</td>
<td>Return bus to synchronous mode if possible</td>
</tr>
<tr>
<td>USCSI_RESET</td>
<td>Reset target</td>
</tr>
<tr>
<td>USCSI_RESET_ALL</td>
<td>Reset all targets</td>
</tr>
<tr>
<td>USCSI_RQENABLE</td>
<td>Enable request sense extensions</td>
</tr>
</tbody>
</table>

#### Interpretation of UsCSI Flags

- **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.
Ioctl Requests

**USCSI_RESET_ALL**

Cause a SCSI Bus Reset on the bus associated with this target.

**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.

**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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>SUNWhea</td>
</tr>
</tbody>
</table>

**SEE ALSO**

`ioctl(2)`, `attributes(5)`, `sd(7D)`, `st(7D)`, ANSI Small Computer System Interface-2 (SCSI-2)

Last modified 5 May 1998
WARNINGS

The \texttt{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 \texttt{uscsi} commands can change the state of the device and confuse the device driver. It is best to use \texttt{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.
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.

One ioctl, VIS_GETIDENTIFIER, 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.

Two other sets of ioctls exist. One set supports mouse tracking via hardware
cursor operations. These are optional, but if a graphics device has hardware
cursor support and implements these ioctls the mouse tracking performance
will be improved.

The other set supports the device being the system console device. These are
optional, but if a graphics device is to be used as the system console device, it
must implement these ioctls.

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.
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];
};

We suggest the name be formed as
<companysymbol><devicetype>. For example, the
cgsix driver returns SUNWcg6.

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 */
The `vis_cursorcmap` structure should contain pointers to two elements, specifying the red, green, and blue values for foreground and background.

**VIS_MOVECURSOR**

These ioctls fetch and move the current cursor position, using the `vis_cursorpos` structure.

**VIS_SETCURSORPOS**

The following set of ioctls are used by graphics drivers that are to be 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  Sets or gets color map entries. The argument is a pointer to a vis_cmap structure which contains the following fields:

```c
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.

```c
struct vis_devinit {
    int version;
    screen_size_t width;
    screen_size_t height;
    screen_size_t linebytes;
    unit_t size;
    int depth;
    short mode;
};
```

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 bits of the device. Currently supported depths are: 1, 4, 8 and 24.
mode is the mode of the device. One of VIS_PIXEL (data to be displayed is in bitmap format) or VIS_TEXT (data to be displayed is in ascii format).

**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_CONS_MODE_CHANGE**
Tells the graphics driver that the framebuffer resolution has been reset by the user program. The framebuffer is expected to reload any state information that it is keeping. The argument to this ioctl is private to the user program and the device driver. That is, the user program may wish to directly change the framebuffer mode and then just use this ioctl to notify the graphics driver or it may pass mode change information along to the graphics driver and have it do the mode change.

**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:

```c
struct vis_conscursor {
    int version;
    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;
};
```

version is set to VIS_CURSOR_VERSION and should be check by the driver. If the version does not match, the driver should reject this ioctl.

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 was 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.
**ioctl Requests**

**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** is whether to display or hide the cursor. It is set to one of: **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:

```c
struct vis_display {
    int version;
    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;
};
```

**version** is set to **VIS_DISPLAY_VERSION** and should be check by the driver. If the version does not match, the driver should reject this ioctl.

**row** and **col** specify the starting row and column to display the data at. 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 rectangle 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 rectangle 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 expect 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:

```c
struct vis_copy {
    int version
    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;
};
```

version is set to VIS_COPY_VERSION and should be check by the driver. If the version does not match, the driver should reject this ioctl.

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
ioctl Requests

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.

The next set of console ioctls are used on systems which don’t have a prom. Normally, standalones use
the system prom to display characters on the system console device. On systems without a prom, standalones use the
kernel drivers to display characters on the system console device. When implementing these ioctls, you can not use any of the
locking primitives or the copy routines from the DDI. Furthermore other DDI services may or may not
work and should be avoided.

VIS_STAND_CONSCURSOR
Should perform the same tasks as VIS_CONSCURSOR except that it must follow the above restrictions.
Takes in as an argument a vis_cursor structure.

VIS_STAND_CONSDISPLAY
Should perform the same tasks as VIS_CONSDISPLAY except that it must follow the above restrictions.
Takes in as an argument a vis_display structure.

VIS_STAND_CONSCOPY
Should perform the same tasks as VIS_CONSCOPY except that it must follow the above restrictions.
Takes in as an argument a vis_copy structure.

Last modified 6 Dec 1995
SunOS 5.7
558
NAME
volfs – Volume Management file system

DESCRIPTION
volfs is the Volume Management file system rooted at root_dir. The default location for root-dir is /vol, but this can be overridden using the -d option of vold (see vold(1M)). This file system is maintained by the Volume Management daemon, vold, and will be considered to be /vol for this description.

Media can be accessed in a logical manner (no association with a particular piece of hardware), or a physical manner (associated with a particular piece of hardware).

Logical names for media are referred to through /vol/dsk and /vol/rdsk. /vol/dsk provides block access to random access devices. /vol/rdsk provides character access to random access devices.

The /vol/rdsk and /vol/dsk directories are mirrors of one another. Any change to one is reflected in the other immediately. The dev_t for a volume will be the same for both the block and character device.

The default permissions for /vol are mode=0555, owner=root, group=sys. The default permissions for /vol/dsk and /vol/rdsk are mode=01777, owner=root, group=sys.

Physical references to media are obtained through /vol/dev. This hierarchy reflects the structure of the /dev name space. The default permissions for all directories in the /vol/dev hierarchy are mode=0555, owner=root, group=sys.

mkdir(2), rmdir(2), unlink(2) (rm), symlink(2) (ln -s), link(2) (ln), and rename(2) (mv) are supported, subject to normal file and directory permissions.

The following system calls are not supported in the /vol filesystem: creat(2), only when creating a file, and mknod(2).

If the media does not contain file systems that can be automatically mounted by rmmount(1M), users can gain access to the media through the following /vol locations.

<table>
<thead>
<tr>
<th>Location</th>
<th>State of Media</th>
</tr>
</thead>
<tbody>
<tr>
<td>/vol/dev/diskette0/unnamed_floppy</td>
<td>formatted unnamed floppy-block device access</td>
</tr>
<tr>
<td>/vol/dev/rdiskette0/unnamed_floppy</td>
<td>formatted unnamed floppy-raw device access</td>
</tr>
<tr>
<td>/vol/dev/diskette0/unlabeled</td>
<td>unlabeled floppy-block device access</td>
</tr>
</tbody>
</table>
For more information on the location of CD-ROM and floppy media, see `System Administration Guide, Volume I` or `rmmount(1M)`.

**Partitions**

Some media supports the concept of a partition. If the label identifies partitions on the media, the name of the media will become a directory with partitions under it. Only valid partitions are represented. Partitions cannot be moved out of a directory.

Example: disk volume ‘foo’ has 3 valid partitions: 0, 2, 5.

```
/vol/dsk/foo/s0, /vol/dsk/foo/s2, /vol/dsk/foo/s5,
/vol/rdsk/foo/s0, /vol/rdsk/foo/s2, /vol/rdsk/foo/s5
```

If a volume is relabeled to reflect different partitions, the name space changes to reflect the new partition layout.

A format program can check to see if there are others with the volume open and not allow the format to occur if it is. Volume Management, however, does not explicitly prevent the rewriting of a label while others have the volume open. If a partition of a volume is open, and the volume is relabeled to remove that partition, it will appear exactly as if the volume were missing. A notify event will be generated and the user may cancel the operation with `volcancel(1)`, if desired.

SEE ALSO

`volcancel(1)`, `volcheck(1)`, `volmissing(1)`, `rmmount(1M)`, `void(1M)`, `rmmount.conf(4)`, `void.conf(4)`

`Solaris 1.x to 2.x Transition Guide System Administration Guide, Volume I`
NAME
vuidmice, vuidm3p, vuidm4p, vuidm5p, vuid2ps2, vuid3ps2 – converts mouse protocol to Firm Events

SYNOPSIS
#include <sys/stream.h>
#include <sys/vuid_event.h>

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. The user will need to change the state to VUID_FIRM_EVENT mode in order to initiate mouse protocol conversion to Firm events. This can be accomplished by the following code:

```c
int format;
format = VUID_FIRM_EVENT;
ioctl(fd, VUIDSFORMAT, &format);
```

The user can also query the state of the STREAMS module by using the VUIDFORMAT option.

```c
int format;
int fd; /* file descriptor */
ioctl(fd, VUIDFORMAT, &format);
if ( format == VUID_NATIVE )
  { /* The state of the module is in raw mode. */
  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 The messages coming in are queued and converted to Firm events.
**STREAMS Modules**

<table>
<thead>
<tr>
<th>M_FLUSH</th>
<th>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.</th>
</tr>
</thead>
</table>

**Write Side Behavior**

<table>
<thead>
<tr>
<th>M_IOCTL</th>
<th>messages sent downstream as a result of an ioctl(2) system call. There are two valid ioctl options processed by the vuidmice modules VUIDGFORMAT and VUIDSFORMAT.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VUIDGFORMAT</td>
<td>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).</td>
</tr>
<tr>
<td>VUIDSFORMAT</td>
<td>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 then 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*).</td>
</tr>
<tr>
<td>M_FLUSH</td>
<td>The write queue of the module is flushed of all its data messages and the message is passed downstream.</td>
</tr>
</tbody>
</table>

**Mouse Configurations**

```
Module\011Protocol Type\011Device
vuidm3p \0113-Byte Protocol \011Microsoft 2 Button Serial Mouse\011 /dev/tty*
vuidm4p \0114-Byte Protocol \011Logitech 3 Button Mouseman \011
```

Last modified 1 Jan 1997

SunOS 5.7 562
/dev/tty*
vuidm5p
\0115-Byte Protocol
\011Logitech 3 Button Bus Mouse\011
/dev/logi

\011Microsoft Bus Mouse\011
/dev/msm
vuid2ps2
\011PS/2 Protocol
\0112 Button PS/2 Compatible Mouse\011
/dev/kdmouse
vuid3ps2
\011PS/2 Protocol
\0113 Button PS/2 Compatible Mouse\011
/dev/kdmouse

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

SEE ALSO attributes(5)
STREAMS Programming Guide
NAME
wscons – workstation console

SYNOPSIS
#include <sys/strredir.h>

DESCRIPTION
The “workstation console” is a device consisting of the combination of the
workstation keyboard and frame buffer, acting in concert to emulate an ASCII
terminal. It includes a redirection facility that allows I/O issued to the
workstation console to be diverted to some other STREAMS device, so that, for
example, window systems can arrange to redirect output that would otherwise
appear directly on the frame buffer, corrupting its appearance.

Redirection
The redirection facility maintains a list of devices that have been named as
redirection targets, through the SRIOSCSREDIR ioctl described below. All
entries but the most recent are inactive; when the currently active entry is
closed, the most recent remaining entry becomes active. The active entry acts
as a proxy for the device being redirected; it handles all read(2), write(2),
ioctl(2), and poll(2) calls issued against the redirectee.

The following two ioctls control the redirection facility. In both cases, fd is a
descrriptor for the device being redirected (that is, the workstation console) and
target is a descriptor for a STREAMS device.

SRIOSCSREDIR Make target be 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.

SPARC ANSI
Standard Terminal
Emulation
On SPARC based systems, the PROM monitor emulates an ANSI X3.64
terminal.

Note: the VT100 also follows the ANSI X3.64 standard but both the Sun and
the VT100 have nonstandard extensions to the ANSI X3.64 standard. The Sun
terminal emulator and the VT100 are not compatible in any true sense.

The Sun console displays 34 lines of 80 ASCII characters per line, with
scrolling, (x, y) cursor addressability, and a number of other control functions.

While the display size is usually 34 by 80, there are instances where it may be
a different size.

- If the display device is not large enough to display 34 lines of text.
- On SPARC based systems, if either screen-#rows or screen-#columns is set
  by the user to a value other than the default of 34 or 80 respectively.
  screen-#rows and screen-#columns are fields stored in
  NVRAM/EEPROM, see eeprom(1M).

Last modified 27 Jan 1998
The Sun console displays a cursor which marks the current line and character position on the screen. ASCII characters between 0x20 (space) and 0x7E (tilde) inclusive are printing characters — when one is written to the Sun 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, not just ASCII. Earlier PROM revisions display characters in the range 0xA0 – 0xFE as spaces.

If the cursor is already at the right edge of the screen, it moves to the first character position on the next line. If the cursor is already at the right edge of the screen on the bottom line, the Line-feed function is performed (see CTRL-J below), which scrolls the screen up by one or more lines or wraps around, before moving the cursor to the first character position on the next line.

The Sun console defines a number of control sequences which may occur in its input. When such a sequence is written to the Sun console, it is not displayed on the screen, but effects some control function as described below, for example, moves the cursor or sets a display mode.

Some of the control sequences consist of a single character. The notation

CTRL-X

for some character X, represents a control character.

Other ANSI control sequences are of the form

```
ESC [ params char
```

Spaces are included only for readability; these characters must occur in the given sequence without the intervening spaces.

**ESC** represents the ASCII escape character (ESC, CTRL-[, 0x1B).

**[** The next character is a left square bracket ‘[’ (0x5B).

**params** are a sequence of zero or more decimal numbers made up of digits between 0 and 9, separated by semicolons.

**char** represents a function character, which is different for each control sequence.

Some examples of syntactically valid escape sequences are (again, ESC represent the single ASCII character ‘Escape’):
Syntactically valid ANSI escape sequences which are not currently interpreted by the Sun console are ignored. Control characters which are not currently interpreted by the Sun console are also ignored.

Each control function requires a specified number of parameters, as noted below. If fewer parameters are supplied, the remaining parameters default to 1, except as noted in the descriptions below.

If more than the required number of parameters is supplied, only the last $n$ are used, where $n$ is the number required by that particular command character. Also, parameters which are omitted or set to zero are reset to the default value of 1 (except as noted below).

Consider, for example, the command character M which requires one parameter. ESC[;M and ESC[0M and ESC[M and ESC[23;15;32;1M are all equivalent to ESC[1M and provide a parameter value of 1. Note: ESC[;5M (interpreted as ‘ESC[5M’) is not equivalent to ESC[5;1M (interpreted as ‘ESC[5;1M’) which is ultimately interpreted as ‘ESC[1M’).

In the syntax descriptions below, parameters are represented as ‘#’ or ‘#1;#2’.

SPARC ANSI Control Functions

The following paragraphs specify the ANSI control functions implemented by the Sun console. Each description gives:

- the control sequence syntax
- the hex equivalent of control characters where applicable
- the control function name and ANSI or Sun abbreviation (if any).
- description of parameters required, if any
- description of the control function
- for functions which set a mode, the initial setting of the mode. The initial settings can be restored with the SUNRESET escape sequence.

CTRL-G (0x7) Bell (BEL)

The Sun Workstation Model 100 and 100U is not equipped with an audible bell. It ‘rings the bell’ by flashing the
entire screen. The window system flashes the window. The screen will also be flashed on current models if the Sun keyboard is not the console input device.

**CTRL-H (0x8) Backspace (BS)**

The cursor moves one position to the left on the current line. If it is already at the left edge of the screen, nothing happens.

**CTRL-I (0x9) Tab (TAB)**

The cursor moves right on the current line to the next tab stop. The tab stops are fixed at every multiple of 8 columns. If the cursor is already at the right edge of the screen, nothing happens; otherwise the cursor moves right a minimum of one and a maximum of eight character positions.

**CTRL-J (0xA) Line-feed (LF)**

The cursor moves down one line, remaining at the same character position on the line. If the cursor is already at the bottom line, the screen either scrolls up or “wraps around” depending on the setting of an internal variable $S$ (initially 1) which can be changed by the ESC[r control sequence. If $S$ is greater than zero, the entire screen (including the cursor) is scrolled up by $S$ lines before executing the line-feed. The top $S$ lines scroll off the screen and are lost. $S$ new blank lines scroll onto the bottom of the screen. After scrolling, the line-feed is executed by moving the cursor down one line.

If $S$ is zero, ‘wrap-around’ mode is entered. ‘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. When any line-feed occurs, the line that the
cursor moves to is cleared. This means that no scrolling occurs.
Wrap-around mode is not implemented in the window system.

On SPARC based systems, the screen scrolls as fast as possible depending on how much data is backed up waiting to be printed. Whenever a scroll must take place 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 any 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\} is found. At that point, the screen is scrolled by N lines (N ⩾ 1) and processing continues. The scanned text is still processed normally to fill in the newly created lines. This results in much faster scrolling with scrolling as long as no escape codes or other control characters are intermixed with the text.

See also the discussion of the ‘Set scrolling’ (ESC[r) control function below.

<table>
<thead>
<tr>
<th>Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL-K (0xB)</td>
<td>Reverse Line-feed</td>
</tr>
<tr>
<td></td>
<td>The cursor moves up one line, remaining at the same character position on the line. If the cursor is already at the top line, nothing happens.</td>
</tr>
<tr>
<td>CTRL-L (0xC)</td>
<td>Form-feed (FF)</td>
</tr>
<tr>
<td></td>
<td>The cursor is positioned to the Home position (upper-left corner) and the entire screen is cleared.</td>
</tr>
<tr>
<td>CTRL-M (0xD)</td>
<td>Return (CR)</td>
</tr>
<tr>
<td></td>
<td>The cursor moves to the leftmost character position on the current line.</td>
</tr>
</tbody>
</table>
### SPARC Escape Sequence Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CTRL-[ (0x1B) Escape (ESC)</strong></td>
<td>This is the escape character. Escape initiates a multi-character control sequence.</td>
</tr>
<tr>
<td><strong>ESC[#@ Insert Character (ICH)</strong></td>
<td>Takes one parameter, # (default 1). Inserts # spaces at the current cursor position. The tail of the current line starting at the current cursor position inclusive is shifted to the right by # character positions to make room for the spaces. The rightmost # character positions shift off the line and are lost. The position of the cursor is unchanged.</td>
</tr>
<tr>
<td><strong>ESC[#A Cursor Up (CUU)</strong></td>
<td>Takes one parameter, # (default 1). Moves the cursor up # lines. If the cursor is fewer than # 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.</td>
</tr>
<tr>
<td><strong>ESC[#B Cursor Down (CUD)</strong></td>
<td>Takes one parameter, # (default 1). Moves the cursor down # lines. If the cursor is fewer than # 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.</td>
</tr>
<tr>
<td><strong>ESC[#C Cursor Forward (CUF)</strong></td>
<td>Takes one parameter, # (default 1). Moves the cursor to the right by # character positions on the current line. If the cursor is fewer than # positions from the right edge of the screen, moves the cursor to the rightmost position on the current line.</td>
</tr>
<tr>
<td><strong>ESC[#D Cursor Backward (CUB)</strong></td>
<td>Takes one parameter, # (default 1). Moves the cursor to the left by # character positions on the current line. If the cursor is fewer than # positions from the left edge of the screen, moves the cursor to the leftmost position on the current line.</td>
</tr>
</tbody>
</table>
| **ESC\[#E Cursor Next Line (CNL)** | Takes one parameter, # (default 1). Positions the cursor at the leftmost character position on the #-th line below the current line. If the current line
is less than \# lines from the bottom of the screen, positions the cursor at the leftmost character position on the bottom line.

**ESC[\#1;\#2f** **Horizontal And Vertical Position (HVP)**

or

**ESC[\#1;\#2H** **Cursor Position (CUP)**

Takes two parameters, \#1 and \#2 (default 1, 1). Moves the cursor to the \#2-th character position on the \#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.

**ESC[J** **Erase in Display (ED)**

Takes no parameters. Erases from the current cursor position inclusive to the end of the screen. In other words, erases from the current cursor position inclusive to the end of the current line and all lines below the current line. The cursor position is unchanged.

**ESC[K** **Erase in Line (EL)**

Takes no parameters. Erases from the current cursor position inclusive to the end of the current line. The cursor position is unchanged.

**ESC[\#L** **Insert Line (IL)**

Takes one parameter, \# (default 1). Makes room for \# new lines starting at the current line by scrolling down by \# lines the portion of the screen from the current line inclusive to the bottom. The \# new lines at the cursor are filled with spaces; the bottom \# lines shift off the bottom of the screen and are lost. The position of the cursor on the screen is unchanged.

**ESC[\#M** **Delete Line (DL)**

Takes one parameter, \# (default 1). Deletes \# lines beginning with the current line. The portion of the screen from the current line inclusive to the bottom is scrolled upward by \# lines. The \# new lines scrolling onto the bottom of the screen are filled with spaces; the \# old lines beginning at the cursor line are deleted. The position of the cursor on the screen is unchanged.

**ESC[\#P** **Delete Character (DCH)**
wscons(7D) Devices

Takes one parameter, # (default 1). Deletes # characters starting with the current cursor position. Shifts to the left by # character positions the tail of the current line from the current cursor position inclusive to the end of the line. Blanks are shifted into the rightmost # character positions. The position of the cursor on the screen is unchanged.

**ESC[#m Select Graphic Rendition (SGR)**

Takes one parameter, # (default 0). Note: 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. Currently 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 # is currently equivalent to 7 and selects the negative image rendition.

**ESC[p Black On White (SUNBOW)**

Takes no parameters. 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) is white-on-black in this mode. This is the initial setting of the screen mode on reset.

**ESC[q White On Black (SUNWOB)**

Takes no parameters. 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) is black-on-white in this mode. The initial setting of the screen mode on reset is the alternative mode, black on white.

**ESC[#r Set scrolling (SUNSCRL)**

Takes one parameter, # (default 0). Sets to # 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. In wrap mode, if a linefeed occurs on the bottom line, the cursor goes to the same character position in the top line of the screen. When any linefeed occurs, the line that the cursor moves to is cleared. This means that no scrolling ever 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.

<table>
<thead>
<tr>
<th>RETURN VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>When there are no errors, the redirection ioctls have return values as described above. Otherwise, they return −1 and set <strong>errno</strong> to indicate the error.</td>
</tr>
<tr>
<td>If the target stream is in an error state, <strong>errno</strong> is set accordingly.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ERRORS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EBADF</strong> target does not denote an open file.</td>
</tr>
<tr>
<td><strong>ENOSTR</strong> target does not denote a STREAMS device.</td>
</tr>
<tr>
<td><strong>EINVAL</strong>(x86 only) <strong>fd</strong> does not denote /dev/console.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FILES</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/wscons the workstation console, accessed by way of the redirection facility</td>
</tr>
<tr>
<td><strong>x86 Only</strong></td>
</tr>
<tr>
<td>/dev/systty</td>
</tr>
<tr>
<td>/dev/syscon</td>
</tr>
<tr>
<td>/dev/console the device that must be opened for the SRIOCREDIR and SRIOCISREDIR ioctls</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SEE ALSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>console(7D)</td>
</tr>
</tbody>
</table>

Last modified 27 Jan 1998

SunOS 5.7

572
WARNINGS
The redirection ioctl block while there is I/O outstanding on the device instance being redirected. Thus, attempting to redirect the workstation console while there is a read outstanding on it will hang until the read completes.

NOTES
On Sun Enterprise 10000 servers the netcon facility supersedes wscons(7D). wscons is useful for systems that do have directly attached consoles, such as frame buffers and keyboards, but it is not useful with the Enterprise 10000 server, which does not. For more information, refer to netcon(1M) in the Sun Enterprise 10000 SSP Reference Manual or cvcd(1M).
NAME xd, xdc – disk driver for Xylogics 7053 SMD Disk Controller

SYNOPSIS

xdc@6d,ee80/xd slave 0: partition
xdc@6d,ee90/xd slave 0: partition
xdc@6d,eea0/xd slave 0: partition
xdc@6d,eeb0/xd slave 0: partition

DESCRIPTION

The driver for Xylogics 7053 devices consists of several components: a controller driver (xdc) and a slave device driver module (xd). Each driver module has an associated configuration file, which lives in the same directory as the driver module. See driver.conf(4) and vme(4) for the interpretation of the contents of these files.

The block files access the disk using the system’s normal buffering mechanism and may be 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 only one I/O operation; therefore raw I/O is considerably more efficient when many words are transmitted. The physical names of the raw files conventionally have ‘,raw’ appended to them. The logical names for the raw files live in the /dev/rdsk directory, as usual.

When using raw I/O, transfer counts should be multiples of 512 bytes (the size of a disk sector). Likewise, when using lseek(2) to specify block offsets from which to perform raw I/O, the logical offset should also be a multiple of 512 bytes.

Partition 0 is normally used for the root file system on a disk, partition 1 as a paging area (for example, swap), and partition 2 for backing 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.

DISK SUPPORT

This driver handles all SMD drives by reading a label from sector 0 of the drive which describes the disk geometry and partitioning.

FILES

/kernel/drv/xdc driver module
/kernel/drv/xd driver module
/kernel/drv/xdc.conf driver configuration file
/kernel/drv/xd.conf driver configuration file
/dev/dsk/c X d Y s Z  block devices, controller X, unit Y, slice Z
/dev/rdsk/c X d Y s Z  raw devices, controller X, unit Y, slice Z

ATTRIBUTES

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>SPARC (Sun-4/200, Sun-4/300, and Sun-4/400 series only)</td>
</tr>
</tbody>
</table>

SEE ALSO

lseek(2), read(2), write(2), driver.conf(4), vme(4), attributes(5), dkio(7I), hdio(7I)

NOTES

In raw I/O read(2) and write(2) truncate file offsets to 512-byte block boundaries, and write(2) scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, read(2), write(2), and lseek(2) should always deal in 512-byte multiples.
**NAME**

xt - driver for Xylogics 472 1/2 inch tape controller

**SYNOPSIS**

```
xt@2d,ee60:[l,m][b][n]
xt@2d,ee68:[l,m][b][n]
```

**DESCRIPTION**

The Xylogics 472 tape controller controls Pertec-interface 1/2" tape drives such as the Fujitsu M2444 and the CDC Keystone III. The xt driver provides a standard tape interface to the device; see `mtio(7I)` for details.

The xt driver supports the character device interface. The driver can be opened with either rewind on close or no rewind on close options. The tape format and options are specified using the device name (see FILES below).

**EOT Handling**

The user will be notified of end of tape (EOT) on write by a 0 byte count returned the first time this is attempted. This write must be retried by the user. Subsequent writes will be successful until the tape winds off the reel. Reading past EOT is transparent to the user.

**IOCTL**

See `mtio(7I)` for a list of ioctls available for tape devices. However, not all devices support all ioctls. The driver returns an ENOTTY error on unsupported ioctls.

1/2" tape devices do not support the tape retension function.

**ERRORS**

- **EACCES** The driver is opened for write access and the tape is write protected.
- **EBUSY** The tape drive is in use by another process. Only one process can use the tape drive at a time.
- **EINVAL** The requested number of bytes for a read operation is less than the actual record length on the tape.
- **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.

**FILES**

- `/kernel/drv/xt` driver module
- `/kernel/drv/xt.conf` driver configuration file

Last modified 1 Jan 1997

SunOS 5.7

576
/dev/rmt/[0-1][l,m][b][n] raw devices
For raw devices l,m specifies the density (low, medium), and 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, the densities are:
l typically 1600 BPI density
m typically 6250 BPI density

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>SPARC (Sun-4/200, Sun-4/300, and Sun-4/400 series only)</td>
</tr>
</tbody>
</table>

SEE ALSO
ioctl(2), driver.conf(4), vme(4), attributes(5), mtio(7I)

BUGS
Record sizes are restricted to an even number of bytes.
The EOT handling for write operation differs from the mtio(7I) specification.
NAME  xy, xyc – disk driver for Xylogics 450 and 451 SMD Disk Controllers

SYNOPSIS  
\[ \text{xc} \text{c@2d,ee40/xy@ slave ,0: partition} \]
\[ \text{xc} \text{c@2d,ee48/xy@ slave ,0: partition} \]

DESCRIPTION  The driver for Xylogics 450/451 devices consists of several components: a controller driver module (xc) and a slave device driver module (xy). Each driver module has an associated configuration file, which lives in the same directory as the driver module. See `driver.conf(4)` and `vme(4)` for the interpretation of the contents of these files.

The block files access the disk using the system’s normal buffering mechanism and may be 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 only one I/O operation; therefore raw I/O is considerably more efficient when many words are transmitted. The physical names of the raw files conventionally have 'raw' appended to them. The logical names for the raw files live in the `/dev/rdsk` directory, as usual.

When using raw I/O, transfer counts should be multiples of 512 bytes (the size of a disk sector). Likewise, when using `lseek(2)` to specify block offsets from which to perform raw I/O, the logical offset should also be a multiple of 512 bytes.

Partition 0 is normally used for the root file system on a disk, partition 1 as a paging area (for example, `swap`), and partition 2 for backing 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.

Due to word ordering differences between the disk controller and Sun computers, user buffers that are used for raw I/O must not begin on odd byte boundaries.

DISK SUPPORT  This driver handles all SMD drives by reading a label from sector 0 of the drive which describes the disk geometry and partitioning.

FILES  
\[ /\text{kernel/drv/xc} \]  driver module  
\[ /\text{kernel/drv/xy} \]  driver module  
\[ /\text{kernel/drv/xc.conf} \]  driver configuration file  
\[ /\text{kernel/drv/xy.conf} \]  driver configuration file  
\[ /\text{dev/dsk/c X d Y s Z} \]  block device, controller X, unit Y, slice Z
/dev/rdsk/c X d Y s Z raw device, controller X, unit Y, slice Z

**ATTRIBUTES**

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

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>SPARC (Sun-4/200, Sun-4/300, and Sun-4/400 series only)</td>
</tr>
</tbody>
</table>

**SEE ALSO**

lseek(2), read(2), write(2), driver.conf(4), vme(4), attributes(5), dkio(7I), hdio(7I)

**NOTES**

In raw I/O read(2) and write(2) truncate file offsets to 512-byte block boundaries, and write(2) scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, read(2), write(2), and lseek(2) should always deal in 512-byte multiples.
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 with `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>
```

**DESCRIPTION**
The Zilog 8530 provides two serial input/output channels that are 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 8530 hardware, together with basic asynchronous communication support. The driver 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/term/a` and `/dev/term/b` are used to access the two serial ports on the CPU board.

**Note:** `/dev/cua/[a-z]`, `/dev/term/[a-z]` and `/dev/tty[a-z]` are valid name space entries. The number of entries used in this name space are machine dependent.

The `/dev/tty` device names only exist if the binary compatibility package is installed. The `/dev/tty` device names are created by the `ucblinks` command. This command is only available via the binary compatibility package.

To allow a single tty line to be connected to a modem and used for both incoming and outgoing calls, a special feature, controlled by the minor device number, is available. 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, after which Carrier Detect will usually drop as well) and carrier is detected again, and a non-blocking open will return an error. Also, 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, for example, `/dev/term/n` (renamed from `/dev/tty`) and...
Devices

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.

**IOCTLS**

The standard set of `termio ioctl()` calls are supported by zs.

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 `TIOCMBRK 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, TIOCMBS, and TIOCMGET ioctl` commands.

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

**ERRORS**

An `open()` will fail if:

**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

Last modified 1 Jan 1997

SunOS 5.7  582
/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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>SPARC</td>
</tr>
</tbody>
</table>

SEE ALSO
cu(1C), tip(1), ucblinks(1B), uucp(1C), ports(1M), ioctl(2), open(2), attributes(5), ldterm(7M), termio(7I), ttcompat(7M), zsh(7D)

Binary Compatibility Guide

DIAGNOSTICS
zs:n: silo overflow. The 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 zshn 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
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 */
```

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.
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:
EINVALAn attempt was made to select an invalid clocking source.
EINVALThe 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:

<table>
<thead>
<tr>
<th>ATTRIBUTE TYPE</th>
<th>ATTRIBUTE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86</td>
</tr>
</tbody>
</table>

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.

zshn: 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.

zshn: 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.

zshn: transmit hung

The transmitter was not successfully restarted after the watchdog timer expired.
Index

Numbers
1/2-inch tape drive  
xt — Xylogics 472, 576
3COM EtherLink III Ethernet device driver — 
elx, 123
3COM EtherLink III PCMCIA Ethernet 
Adapter — pcelx, 316
4BSD compatibility module — ttcompat, 536
24-bit UPA color frame buffer and graphics 
accelerator — ffb, 159
450 SMD Disk driver — xy, 578
451 SMD Disk driver — xy, 578
472 1/2-inch tape drive — xt, 576
7053 SMD Disk driver — xd, 574

A
Adaptec 154x ISA host bus adapter — aha, 15
Adaptec 174x EISA host bus adapter — 
aha, 122
Address Resolution Protocol, see ARP, 
adp — low-level module for controllers based 
on Adaptec AIC-7870P and 
AIC-7880P SCSI chips, 13
aha — low-level module for Adaptec 154x ISA 
host bus adapter, 15
aie — low-level module for Adaptec AIC-6360 
based ISA host bus 
adapters, 17
Am7990 (LANCICE) Ethernet device driver 
— le, 252
— lebuffer, 252
— ledma, 252
Am79C940 (MACE) Ethernet device driver — 
qe, 356, 359
AMD PCnet Ethernet controller device driver 
— pcn, 325
an ISO specific module that supports SCSA 
interface 
an ISO, 183
ANSI Layered Console Driver — ltem, 270
ANSI standard terminal emulation — 
w cons, 564
arp — Address Resolution Protocol, 19
SIOCDARP — delete arp entry, 20
SIOCGARP — get arp entry, 20
SIOCSARP — set arp entry, 20
asy — asynchronous serial port driver, 22
asynchronous serial port driver — asy, 22
AT attachment disk driver — ata, 25
ata — AT attachment disk driver, 25
audio — audio device interface, 27
audio device 
Sound Blaster 16/Pro/AWE32 — 
sbpro, 381
audioamd — telephone quality audio 
device, 38
audiocs — Crystal Semiconductor 4231 audio 
Interface, 40
Audio Data Formats for the Multimedia 
4231 Codec, 41
Audio Interfaces, 40
Audio Ports, 41
Audio Status Change Notification, 42
Sample Granularity, 42
connections, unique stream
line discipline — connld, 81
connld — line discipline for unique connections, 81
console — STREAMS-based console interface, 83
converts mouse protocol to Firm Events — vuidmice, 561
  vuid2ps2, 561
  vuid3ps2, 561
  vuidm3p, 561
  vuidm4p, 561
  vuidm5p, 561
core memory
  image — mem, 273
cpqncr — low-level module for Compaq 825 and 875 Controllers, 84
cpr — suspend and resume module, 86
Crystal Semiconductor 4231 audio Interface — audios, 40
csa — low-level module for Compaq SMART Array Controller, 89
cvc — virtual console driver, 90
cvcredirect — virtual console redirection driver, 91

D
D-LINK Ethernet controller device driver — dnet, 111
Data Link Provider Interface
  — dlpi, 110
dbri — ISDN and audio interface, 92
  Audio Data Formats for BRI Interfaces, 94
  Audio Data Formats for the Multimedia Codec/SpeakerBox, 93
Audio Interfaces, 92
  Audio Ports, 95
Audio Status Change Notification, 95
ISDN Interfaces, 93
  Sample Granularity, 95
delete arp entry ioctl — SIOCDDARP, 20
device driver for Compaq NIC — cnft, 77
device interface
  Microsoft Bus Mouse — msm, 283
device
  cgfourteen, 71
disk control operations — dkio, 100
disk driver
    fd — floppy, 148
    Xylogics — xd, 574, 578
disk quotas — quotactl(), 365
display — system console display, 99
dkio — disk control operations, 100
DKIOCEJECT — disk eject, 100
DKIOCGAPART — get full disk partition table, 100
DKIOCGGEOM — get disk geometry, 100
DKIOCGVTOC — get volume table of contents (vtoc), 100
DKIOCINFO — get disk controller info, 100
DKIOCSAPART — set disk partition info, 100
DKIOCSGEMO — set disk geometry, 100
DKIOCSVTOC — set volume table of contents (vtoc), 100
dlpi — Data Link Provider Interface, 110
dnet — DEC 21040/21140-based Ethernet Controllers, 111
DOS
    DOS formatted file system — pcfs, 317
dpt — DPT 2011, 2012, 2021, 2022, 2122, 2024, 2124, 3121, 3222, and 3224 controllers, 113
driver for parallel port — lp, 268
driver for SPARC Storage Array disk devices — ssd, 423
drivers for floppy disks and floppy disk controllers — fd, 148
driver for SCSI disk and (ATAPI/SCSI) CD-ROM devices — sd, 384
    fdc, 148
    SCSI tape devices — st, 428
Dual Basic Rate ISDN and audio Interface — dbri, 92

E

bpp — bi-directional parallel port, 114
eeapro — EtherExpress-Pro Ethernet device driver, 120
eha — low-level module for Adaptec 174x EISA host bus adapter, 122
ehx — 3COM ETHERLINK III Ethernet device driver, 123
ehx Primitives, 123
esa — low-level module for Adaptec 7770 based SCSI controllers, 125
esp — ESP SCSI Host Bus Adapter Driver, 126
ESP SCSI Host Bus Adapter Driver — esp, 126
EtherExpress 16 Ethernet device driver, Intel — iee, 186
EtherExpress-Pro Ethernet device driver, Intel — eepro, 120
Ethernet device driver
    SMC 3032/EISA dual-channel Ethernet device driver — smce, 409
    SMC 8003/8013/8216/8416 Ethernet device driver — smc, 407
    SMC Elite32 Ultra (8232) Ethernet device driver — smceu, 413
    SMC Ether100 (9232) Ethernet device driver — smcf, 415

F

fas — FAS SCSI Host Bus Adapter Driver, 135
FAS SCSI Host Bus Adapter Driver — fas, 135
fbio — frame buffer control operations, 146
    fd — drivers for floppy disks and floppy disk controllers, 148
    fdc — drivers for floppy disks and floppy disk controllers, 148
FDGETCHANGE — get status of disk changed, 154
    fdio — disk control operations, 154
FDIOGCHAR — get floppy characteristics, 154
FDIOGCHAR — set floppy characteristics, 154
FDKEJECT — eject floppy, 154
ffb — 24-bit UPA color frame buffer and graphics accelerator, 159
file system
    quotactl() — disk quotas, 365
flashpt — low-level module for Mylex/BusLogic host bus adapters, 160
    Supported BusLogic Adapters, 160
floppy disk driver — fd, 148
floppy disk control operations — fdio, 154
frame buffer
    black and white frame buffer — bwtwo, 56
frame buffer control operations
G

general properties of Internet Protocol network interfaces
   — if, 195
   — if_tcp, 195
general terminal interface — termio, 487
Generic LAN Driver — gld, 161
gld — Generic LAN Driver, 161
gld and Ethernet V2 and 802.3, 161
gld and Style 1 and 2 Providers, 162
Implemented DLPI Primitives, 162
glm — GLM SCSI Host Bus Adapter Driver, 165
Driver Configuration, 165
GLM SCSI Host Bus Adapter Driver — glm, 165

H

hdio — SMD and IPI disk control operations, 171
High Sierra filesystem, see hsfs,
hme — SUNW,hme Fast-Ethernet device driver, 173
   hme Primitives, 175
hsfs
   filesystem — hsfs, 178

I

I/O
   data link provider interface — dlpi, 110
   extended terminal interface —
      termio, 511
   ioctlts that operate directly on sockets —
      sockio, 422
   STREAMS ioctl commands —
      streamio, 457
IBM 16/4 Token Ring Network Adapter
device driver — tr, 531
icmp — Internet Control Message Protocol, 184
iee — EtherExpress 16 Ethernet device
driver, 186
ieef — Intel EtherExpress Flash32/82596
   Ethernet device driver
   ieef and DLPI, 188
ieef — Intel EtherExpress Flash32/82596
   Ethernet device driver, 188
if — general properties of Internet Protocol
   network interfaces, 195
if_tcp — general properties of Internet Protocol
   network interfaces, 195
   Application Programming Interface, 195
inet — Internet protocol family, 199
Intel D100 Ethernet device driver — iprb, 207
Intel EtherExpress 16 Ethernet device driver —
   iee, 186
Intel EtherExpress Flash32/82596 Ethernet
device driver — ieef, 188
Intel EtherExpress-Pro Ethernet device driver
   — eepro, 120
Intel i82365SL PC Card Interface Controller —
   pcic, 322
Intel Ethernet device driver, Intel — iprb, 207
Internet Control Message Protocol —
   icmp, 184, 202, 479, 545
Internet Protocol
to Ethernet addresses — arp, 19
Internet protocol family — inet, 199
Internet Protocol network interfaces
general properties — if_tcp, 195
ioctlts for sockets
   SIOCDARP — delete arp entry, 20
   SIOCGARP — get arp entry, 20
   SIOCSARP — set arp entry, 20
ioctlts for disks
   DKIOCEJECT — disk eject, 100
   DKIOCGPART — get full disk partition
table, 100
   DKIOCGGEOM — get disk geometry, 100
   DKIOCGVTOC — get volume table of
      contents (vtoc), 100
   DKIOCINFO — get disk controller
      info, 100
   DKIOCSAPART — set disk partition
      info, 100
   DKIOCSGGEOM — set disk geometry, 100
   DKIOCSVTOC — set volume table of
      contents (vtoc), 100
ioctlts for floppy
   FDEJECT — eject floppy, 154

Index-591

man Pages(7):Device and Network Interfaces ☼ October 1998
FDGETCHAGE — get status of disk changed, 154
FDIOCHAR — get floppy characteristics, 154
ioctl’s for terminals
TIOCSPKT — set/clear packet mode (pty), 353
TIOCREMOTE — remote input editing, 354
TIOCSTART — start output (like control-Q), 353
TIOCSTOP — stop output (like control-S), 353
ioctl’s for sockets
SIOCADDRT — add route, 375
SIOCDelRT — delete route, 375
ip — Internet Protocol, 202
ipd — STREAMS modules and drivers for the Point-to-Point Protocol, 345
ipdcm — STREAMS modules and drivers for the Point-to-Point Protocol, 345
ipdptp — STREAMS modules and drivers for the Point-to-Point Protocol, 345
iprb — D100 Ethernet device driver, 207
isdnio — generic ISDN interface, 209
ISO 9660 — ISO 9660 CD-ROM filesystem — hsfs, 178
isp — ISP SCSI Host Bus Adapter Driver, 225
ISP SCSI Host Bus Adapter Driver — isp, 225
K
kb — keyboard STREAMS module, 232
Keyoed Compatibility Mode, 238
Keyboard Translation Mode, 232
Keyboard Translation-Table Entries, 232
kdmouse — built-in mouse device interface, 243
kernel lock statistics driver — lockstat, 260
kernel packet forwarding database — route, 369
kernel statistics driver — kstat, 245
kernel symbols — ksysms, 246
keyboard — system console keyboard, 244
keyboard STREAMS module — kb, 232
kstat — kernel statistics driver, 245
kysms — kernel symbols, 246
L
LAN support module — gld, 161
ldterm — line discipline for STREAMS terminal module, 248
le — Am7990 (LANCE) Ethernet device driver, 252
lebuffer — Am7990 (LANCE) Ethernet device driver, 252
ledma — Am7990 (LANCE) Ethernet device driver, 252
line discipline for unique stream connections — conln, 81
llc1 — Logical Link Control Protocol Class 1 Driver, 257
lockstat — kernel lock statistics driver, 260
log — interface to STREAMS error logging and event tracing, 263
logi — LOGITECH bus mouse device interface, 267
Logical Link Control Protocol Class 1 Driver — llc1, 257
Logical Link Control Protocol Class 1 Driver — llc1, 257
LOGITECH Bus Mouse device interface — logi, 267
loopback file system — lofs, 261
loopback transport providers
— ticls, 517
— ticots, 517
— ticotsord, 517
low-level module
Adaptec 154x ISA host bus adapter — aha, 15
Adaptec 174x EISA host bus adapter — eha, 122
Mylex DAC960E
DAC960P/PD/PD-Ultra/PL host bus adapter series — mlx, 280
low-level module for Adaptec 7770 based SCSI controllers — esa, 125
low-level module for Adaptec AIC-6360 based ISA host bus adapters — aic, 17
Index-592
low-level module for Compaq 825 and Compaq 875 Controllers — cpqncr, 84
low-level module for Compaq SMART Array Controller — csa, 89
low-level module for controllers based on Adaptec AIC-7870P and AIC-7880P SCSI chips — adp, 13
low-level module for Mylex/BusLogic host bus adapters — flashpt, 160
low-level module for NCR 53C710, 53C810, 53C815, 53C820, and 53C825 host bus adapters — ncrs, 299
low-level module for the AMD PCscsi, PCscsi II, PCnet-SCSI, and Qlogic QLA510 PCI-to-SCSI bus adapters — pcscki, 328
lp — driver for parallel port, 268
item — ANSI Layered Console Driver, 270

M
m64 — PCI low-range graphics accelerator with color memory frame buffer, 271
magnetic tape interface
— mtio, 285
mem— image of core memory, 273
memory based filesystem — tmpfs, 528
memory, core
— image — mem, 273
memory, zeroed unnamed
— source — zero, 580
mhd — multihost disk control operations, 274
— Automatic Probe Function, 275
— Failfast, 275
mic — Multi-interface Chip driver, 277
Microsoft Bus Mouse device interface — msm, 283
mlx — low-level module for Mylex DAC960E DAC960P/PD/PD-Ultra/PL host bus adapter series, 280
Access to Ready/Standby Drives, 280
Board Configuration and Auto Configuration, 280
Configuration Tips, 280
Hot Plugging, 281

N
crs — low-level module for NCR 53C710, 53C810, 53C820, and 53C825 host bus adapters, 299
NE2000, NE2000plus Ethernet device driver, Novell — nei, 304
NE3200 Ethernet device driver, Novell — nee, 302
nee — NE3200 Ethernet device driver, 302
nei — NE2000, NE2000plus Ethernet device driver, 304
Netflex-2 Ethernet device driver, Compaq — nfe, 306
network packet routing device — routing, 375
nei — Netflex-2 Ethernet device driver, 306
Novell NE2000, NE2000plus Ethernet device driver — nei, 304
Novell NE3200 Ethernet device driver — nee, 302
null — null file, 308

O
openprom — PROM monitor configuration interface, 309

P
packet routing device — routing, 375
packet routing ioctls
   SIOCADDRT — add route, 375
   SIOCDELRT — delete route, 375
parallel port, bi-directional — bpp, 45, 114
driver for parallel port — lp, 268
pcata — PCMCIA ATA card device driver, 314
pcelx — 3COM EtherLink III PCMCIA Ethernet Adapter, 316
pcfs — DOS formatted file system, 317
pcic — Intel i82365SL PC Card Interface Controller, 322
pcck — STREAMS Packet Mode module, 323
PCMCIA ATA card device driver — pcata, 314
PCMCIA memory card nexus driver — pcmem, 324
PCMCIA RAM memory card device driver — pcram, 327
PCMCIA serial card device driver — pcser, 329
pcmem — PCMCIA memory card nexus driver, 324
pcn — AMD PCnet Ethernet controller device driver, 325
pcram — PCMCIA RAM memory card device driver, 327
pcscsi — low-level module for the AMD
   PCscsi, PCscsi II, PCnet-SCSI, and Qlogic QLA510
   PCI-to-SCSI bus adapters, 328
pcser — PCMCIA serial card device driver, 329
pe — Xircom Pocket Ethernet device driver, 330
pfmod — STREAMS packet filter module, 332
pipemod — STREAMS pipe flushing module, 336
Platform Management Chip driver — pmc, 343
Platform Specific Module (PSM) for Tricord
   Systems Enterprise Server Models ES3000, ES4000 and
   ES5000 — tpf, 530
pln — SPARC Storage Array SCSI Host Bus
   adapter driver, 337
PLN SCSI Host Bus Adapter driver — pln, 337
pm — Power Management Driver, 339
pmc — Platform Management Chip driver, 343
Point-to-Point Protocol
   — ipdptp, 345
   — ppp, 345
   — ppp_diag, 345
Power Management Driver — pm, 339
ppp — STREAMS modules and drivers for the Point-to-Point Protocol, 345
   Operation, 345
   ppp_diag — STREAMS modules and drivers for the Point-to-Point Protocol, 345
PROM
   monitor configuration interface — openprom, 309
Pseudo Terminal Emulation module,
   STREAMS — ptem, 347
   pseudo-terminal driver — pty, 352
   ptem — STREAMS Pseudo Terminal Emulation module, 347
ptm — STREAMS Buffer module, 348
pts — STREAMS pseudo-tty slave driver, 350
pty — pseudo-terminal driver, 352
pty — pseudo-terminal driver, 352
Q
   qe — Am79C940 (MACE) Ethernet device driver, 356
   qec — Am79C940 (MACE) Ethernet device driver, 359
   qfe — SUNW,qfe Quad Fast-Ethernet device driver, 360
   qfe Primitives, 362
   quotactl() — disk quotas, 365
R
   remote input editing ioctl —
      TIOCREMOTE, 354
   rms_smt — Rockwell Station Management driver, 368
   Rockwell 2200 SNAP Streams Driver —
      sxp, 476
   Rockwell Station Management driver —
      rms_smt, 368
   route — kernel packet forwarding database, 369
      Messages, 370

Index-594
Index-595

man Pages(7): Device and Network Interfaces  •  October 1998

r and networking —
local network packet routing, 375
r and networking ioctls
   SIOCADDRT — add route, 375
   SIOCDELRT — delete route, 375

S

sbpro — Creative Labs Sound Blaster audio
device, 381
SCSI disk devices
driver — sd, 384
SCSI enclosure services device driver —
   ses, 399
SCSI tape devices
driver — st, 428
sd — driver for SCSI disk and (ATAPI/SCSI)
   CD-ROM devices, 384
se — Siemens 82532 ESCC serial
   communications driver, 391
serial communications driver — zs, 581
Serial Optical Controller device driver —
   soc, 417
Serial Optical Controller for Fibre Channel
   Arbitrated Loop (SOC+) device driver —
   socal, 419
Serial Parallel Communications driver for
   SBus — stc, 442
ses — SCSI enclosure services device
driver, 399
set/clear
   packet mode (pty) ioctl — TIOCPKT, 353
zsh — On-board serial HDLC interface, 395
Siemens 82532 ESCC serial communications
driver — se, 391
SIOCDARP — delete arp entry, 20
SIOCGARP — get arp entry, 20
SIOCSARP — set arp entry, 20
smartii — Compaq Smart-2 EISA/PCI and
   Smart-2SL PCI Array
   Controller driver, 405
SMC Ethernet device drivers
   smc — SMC 8003/8013/8216/8416
   Ethernet device driver, 407
   smce — SMC 3032/EISA dual-channel
   Ethernet device driver, 409
   smceu — SMC Elite32 Ultra (8232)
   Ethernet device driver, 413
   smcf — SMC Ether100 (9232) Ethernet
device driver, 415
SMC EtherPower 8432BT Ethernet controller
device driver — dnet, 111
SMD and IPI disk control operations —
   hdio, 171
SMD disk controller
   Xylogics 7053 — xd, 574, 578
soc — Serial Optical Controller Device
   Driver, 417
socal — Serial Optical Controller for Fibre
   Channel Arbitrated Loop
   (SOC+) device driver, 419
sockio — ioctls that operate directly on
   sockets, 422
sockets
   ioctls that operate directly — sockio, 422
Solaris VISUAL I/O control operations, 552
Sound Blaster 16/Pro/AWE32 audio devices
   — sbpro, 381
SPARC storage Array
disk devices driver — ssd, 423
   SCSI Host Bus Adapter driver — pln, 337
ssd — driver for SPARC Storage Array disk
   devices, 423
st — driver for SCSI tape devices, 428
start output (like control-Q) ioctl —
   TIOCSTART, 353
stc — Serial Parallel Communications driver
   for SBus, 442
stop output (like control-S) ioctl —
   TIOCuTSTOP, 353
STP 4020 PCMCIA Adapter
   STP 4020 PCMCIA Adapter, 456
stp4020 — STP 4020 PCMCIA Adapter, 456
STREAMS
   console interface — console, 83
   interface to error logging — log, 263
   interface to event tracing — log, 263
   line discipline for unique stream
   connections — connd, 81
   loopback transport providers — ticlts,
   ticots, ticotsord, 517
   On-board serial HDLC interface —
   se_hd1c, 395, 584
   standard terminal line discipline module
   — ldterm, 248
Transport Interface cooperating module — timod, 520
Transport Interface read/write interface module — tirdwr, 522
V7, 4BSD, XENIX compatibility module — ttcompat, 536
STREAMS Administrative Driver — sad, 377
STREAMS Buffer module — bufmod, 51
STREAMS Buffer module — ptm, 348
STREAMS ioctl commands — streamio, 457
STREAMS module
SunButtons and SunDials — bd, 43
 STREAMS modules and drivers for the Point-to-Point Protocol
— ipd, 345
— ipdcm, 345
— ipdptp, 345
— ppp, 345
— ppp.diag, 345
STREAMS Packet Filter Module — pfmod, 332
STREAMS Packet Mode module — pckt, 323
STREAMS pipe flushing module — pipemod, 336
STREAMS Pseudo Terminal Emulation module — ptem, 347
STREAMS pseudo-tty slave driver — pts, 350
SunButtons and SunDials STREAMS module — bd, 43
SUNW,hme Fast-Ethernet device driver — hme, 173
SUNW,qfe Quad Fast-Ethernet device driver — qfe, 360
suspend and resume module — cpr, 86
sxp — Rockwell 2200 SNAP Streams Driver, 476
system console display — display, 99
system console keyboard — keyboard, 244

T
tape drive, 1/2-inch
— xt — Xylogics 472, 576
tape interface — mt, 284
tape, magnetic interface
— mtio, 285
tcp — Internet Transmission Control Protocol, 479
tcx — Sun low-range graphics accelerator with color memory frame buffer, 484
terminal emulation, ANSI — wscons, 564
terminal interface
— termio, 487
terminal interface, extended
— termiov, 511
terminal parameters — termio, 511
terminal, standard STREAMS line discipline module — ldterm, 248
termio — general terminal interface, 487
Canonical mode input processing, 488
Comparison of the different cases of MIN, TIME interaction, 490
Control Modes, 498
Default values, 506
Input modes, 494
Local modes, 502
Minimum and Timeout, 505
Modem disconnect, 493
Modem lines, 506
Non-canonical mode input processing, 489
Output modes, 496
Special Characters, 491
Terminal parameters, 494
Terminal size, 505
Termio structure, 505
Writing characters, 491
termiov — extended general terminal interface, 511
ticlt — loopback transport provider, 517
ticots — loopback transport provider, 517
ticotsord — loopback transport provider, 517
timod — Transport Interface cooperating module, 520, 522
TIOCPKT — set/clear packet mode (pty), 353
TIOCREMOTE — remote input editing, 354
TIOCSTART — start output (like control-Q), 353
TIOCSTOP — stop output (like control-S), 353
tmpfs — memory based filesystem, 528
tpf — Platform Specific Module (PSM) for Tricord Systems Enterprise Server Models ES3000, ES4000 and ES5000, 530
tr — IBM 16/4 token ring network device driver, 531
Transport Interface cooperating STREAMS module — timod, 520
Transport Interface read/write interface STREAMS module — timod, 522
Trantor — Trantor T348 Parallel SCSI host bus adapter, 535
Trantor T348 Parallel SCSI host bus adapter — trantor, 535
ttcompat — V7, 4BSD and XENIX STREAMS compatibility module, 536
tty — controlling terminal interface, 544

U
udp — Internet User Datagram Protocol, 545
unnamed zeroed memory source — zero, 580
uscsi — user SCSI command interface, 547
user SCSI command interface — uscsi, 547

V
V7 compatibility module — ttcompat, 536
virtual console driver — cvc, 90
virtual console redirection driver — cvcredir, 91
volfs — Volume Management file system, 559
Volume Management file system — volfs, 559
vuid2ps2 — converts mouse protocol to Firm Events, 561
vuid3ps2 — converts mouse protocol to Firm Events, 561
vuidm3p — converts mouse protocol to Firm Events, 561
vuidm4p — converts mouse protocol to Firm Events, 561
vuidm5p — converts mouse protocol to Firm Events, 561
vuidmice — converts mouse protocol to Firm Events, 561

W
workstation console — wscons, 564

X
xd — Xylogics SMD Disk driver, 574
XENIX compatibility module — ttcompat, 536
xt — Xylogics 472 1/2-inch tape drive, 576
xy — Xylogics SMD Disk driver, 578
Xylogics SMD Disk driver — xd, 574, 578
Xylogics 472 1/2-inch tape drive — xt, 576

Z
zero — source of zeroes, 580
Zilog 8530 SCC serial communications driver — zs, 581
zs — zilog 8530 SCC serial communications driver, 581
zsh — On-board serial HDLC interface, 584